

Titre: More Controllable Conditional Generative Models of Video and Data
Title: Structures for Design

Auteur: Zhi Hao Luo
Author:

Date: 2024

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Luo, Z. H. (2024). More Controllable Conditional Generative Models of Video and
Data Structures for Design [Thèse de doctorat, Polytechnique Montréal].
Citation: PolyPublie. <https://publications.polymtl.ca/62499/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/62499/>
PolyPublie URL:

**Directeurs de
recherche:** Christopher J. Pal
Advisors:

Programme: Génie informatique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**More Controllable Conditional Generative Models of Video and Data
Structures for Design**

ZHI HAO LUO

Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Novembre 2024

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

**More Controllable Conditional Generative Models of Video and Data
Structures for Design**

présentée par **Zhi Hao LUO**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Sarath Chandar ANBIL PARTHIPAN, président

Christopher J. PAL, membre et directeur de recherche

Aishwarya AGRAWAL, membre

David MEGER, membre externe

DEDICATION

To the nights weighed down by self-doubt and the mornings that rose with renewed resolve.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest appreciation for my partner in crime, Geya, for always being here no matter what. You kept me going and you gave me the final push I needed to get to the finish line.

Thanks to Christopher Pal for taking me on board and sticking with me despite the fruitless effort we exerted on various projects. Thank you for the directions and encouragement you offered when I needed them the most. And thank you for all the help you gave me in this last stretch.

Thanks to my parents for supporting me through this meandering journey and for always believing in me regardless of the circumstances. You allowed me to keep my eyes on the road and never needed to worry about any distraction.

A sincere thank you to Chris Beckham and Sandeep for your help when I was stuck. They made a big difference.

Finally, thank you for my friends whose discord chatter kept me grounded and gave me the balance I needed to not fall on this long journey.

RÉSUMÉ

Les modèles génératifs d’images et de texte ont captivé l’attention du public et deviennent de plus en plus importants et pertinents dans nos vies. Alors que les modèles génératifs deviennent encore plus pertinents pour résoudre des problèmes du monde réel, il devient plus important de comprendre comment contrôler ce qu’ils génèrent. Dans cette thèse, nous investiguons différentes méthodes de conditionnement et de contrôle pour les modèles génératifs de vidéo, de langage et de structures de données représentant les designs de plans de sol. Nous sommes particulièrement intéressés par le contrôle fin, où l’information au niveau pixel est utilisée pour la génération de vidéo conditionnelle et où différentes contraintes quantitatives sont fournies à un générateur de designs architecturaux. Cette thèse est organisée en trois axes de recherche.

Dans le premier axe de travail, nous nous concentrons sur la génération de vidéo conditionnelle basée sur les CNN. Les modèles de génération de vidéo CNN actuels emploient une approche par blocs, ce qui signifie que les frames sont générées par blocs plutôt que une à la fois. Le bloc généré est ensuite utilisé comme conditionnement pour générer le bloc de frames suivant. Nous proposons une nouvelle façon de faire le conditionnement par bloc en le déplaçant dans l’espace de représentation latente plutôt que dans l’espace pixel pour réduire l’erreur cumulative. En outre, nous introduisons un mécanisme de mémoire ainsi qu’un réseau de mise à jour de mémoire novateur pour améliorer la qualité de la génération de séquences longues.

Dans le deuxième axe de travail, nous nous concentrons sur la génération de vidéo contrôlable basée sur la diffusion. Nous introduisons un réseau de génération de vidéo contrôlé par des boîtes de délimitation. Plus spécifiquement, étant donné le premier frame de la vidéo, ainsi que la représentation graphique des boîtes de délimitation des premier et dernier frames, notre réseau de prédiction de boîtes de délimitation peut échantillonner des séquences de boîtes de délimitation intermédiaires. Ensuite, notre réseau de générateur de vidéo utilise cette séquence de boîtes de délimitation pour générer une vidéo réaliste qui respecte strictement les positions de boîtes de délimitation données.

Dans le troisième axe de travail, nous nous concentrons sur la création de plans de sol, formulée comme un problème de génération de langage conditionnel. Les modèles de génération de plans de sol actuels se concentrent principalement sur une approche basée sur les images et rarement tiennent compte des critères numériques. Nous proposons de représenter à la fois les désirs de l’utilisateur et les plans de sol image à l’aide de structures de données. Ensuite,

nous utilisons ces structures de données pour affiner un grand modèle de langage (LLM) pour transformer les désirs et préférences de l'utilisateur partiellement spécifiés en plans de sol qui peuvent être rendus comme des images. Nous montrons que avec cette approche, nos LLM pour la génération de plans de sol peuvent générer des représentations métavalides de plans de sol au niveau des structures de données tout en respectant différentes spécifications quantitatives.

Cette thèse adopte une approche de thèse classique puisque les travaux présentés dans cette thèse sont pour la plupart en cours de révision. Même s'il ne s'agit pas d'une thèse par article, des versions arXiv des contributions sont disponibles.

ABSTRACT

Generative models for images and text have captured the public’s attention and are becoming increasingly important and relevant to our lives. As generative models become even more relevant to real world problems it becomes more important to understand how to control what they generate. In this thesis we investigate different conditioning and control methods for generative models of video, language and data structures representing the designs of floor plans. We are particularly interested in fine grained control, where pixel level information is used for conditional video generation and different types of quantitative constraints are provided to a generator of architectural designs. This thesis is organized into three lines of research.

In the first line of work we focus on CNN-based conditional video generation. Current state of the art video generation CNN models employ a block-to-block approach, meaning frames are generated in blocks instead of one by one. The generated block is then used as conditioning to generate the next block of frames. We propose a novel way to do block conditioning by moving it into the latent representation space instead of the pixel space to reduce cumulative error. In addition, we introduce a memory mechanism as well as a novel memory update network to improve the quality of long sequence generation.

In the second line of work we focus on diffusion based controllable video generation. We introduce a bounding box controlled video generation network. Specifically, given the first frame of the video, along with the image rendering of bounding boxes of the first and last frames, our bounding box predictor network is able to sample sequences of intermediate bounding boxes. Then, our video generator network makes use of this bounding box sequence, and generates a realistic video that strictly adheres to the given bounding box positions.

In the third line of work we focus on floorplan creation, formulated as a conditional language generation problem. Current state of the art floorplan generation models mostly focus on an image-based approach, and rarely accommodate for numerical criteria. We propose to represent both user desires and image floorplans using data structures. In turn we use these data structures to fine tune a large language model (LLM) to transform partially specified user desires and preferences into floor plans that can be rendered as images. We show that with this approach our LLMs for floorplan generation can generate valid meta-representations of floorplans at a data structure level while also respecting different types of quantitative specifications.

This thesis takes a classic thesis approach as the work presented in this thesis are mostly

under review. Even though this is not a thesis by article, arXiv versions of the contributions are available.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vii
TABLE OF CONTENTS	ix
LIST OF TABLES	xiii
LIST OF FIGURES	xvi
LIST OF SYMBOLS AND ACRONYMS	xix
LIST OF APPENDICES	xx
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	1
1.2 Intelligence Through Expression	1
1.3 Generative Models	3
1.4 Controlled Generations	4
1.5 Outline	5
CHAPTER 2 BACKGROUND AND LITERATURE REVIEW	10
2.1 Early Autoregressive Models	10
2.1.1 Recurrent Neural Networks (RNNs)	10
2.1.2 Long Short-Term Memory Networks (LSTMs)	11
2.2 Transformers	12
2.2.1 Attention	12
2.2.2 Encoder and Decoder Architecture	15
2.2.3 Decoder-only Transformers	17
2.2.4 Controllable Text Generation	17
2.2.5 Efficient Fine-Tuning	18
2.3 Convolutional Neural Network	20

2.4	U-Nets	21
2.4.1	3D U-Nets	23
2.5	Diffusion Models	23
2.5.1	Forward Diffusion Process	23
2.5.2	Reverse Denoising Process	24
2.5.3	Stable Diffusion	25
CHAPTER 3 VIDEO PREDICTION VIA SPATIO TEMPORAL EXTERNAL AT-		
	TENTION AND MEMORY	27
3.1	Introduction	28
3.2	Related Work	29
3.3	Method	30
3.3.1	Model Architecture	30
3.3.2	Feature Recycling	36
3.3.3	Training Objectives	38
3.4	Experiments	39
3.4.1	Datasets	39
3.4.2	Pretraining	41
3.4.3	Implementation Details	41
3.4.4	Result Analysis	41
3.5	Ablation Studies	43
3.5.1	Feature vs. Frame Recycling in ViP-STEAM	43
3.5.2	Effect of Block Memory’s Updates	44
3.5.3	Effect of the STEAM block	44
3.5.4	Effect of Different Loss Functions	44
3.6	Conclusion	46
CHAPTER 4 CTRL-V: HIGHER FIDELITY VIDEO GENERATION WITH BOUNDING-		
	BOX CONTROLLED OBJECT MOTION	47
4.1	Introduction	47
4.2	Preliminaries	49
4.2.1	Preliminaries	49
4.3	Related Work	50
4.3.1	Video latent diffusion models	50
4.3.2	Conditional Video Diffusion	50
4.3.3	Modes of Control	50
4.3.4	Bounding Box Based Models	51

4.3.5	Vehicle Oriented Generative Models	51
4.4	Method	52
4.4.1	Overview of our Method: Ctrl-V	52
4.4.2	Ctrl-V: BBox Predictor	52
4.4.3	Ctrl-V: Box2Video	54
4.5	Experiments and Ablation Studies	54
4.5.1	Datasets	55
4.5.2	Training Configurations	55
4.5.3	Metrics	55
4.5.4	Generation Quality	58
4.5.5	BBox Predictor: Quantitative Evaluation	59
4.5.6	Box2Video: Motion Control Evaluation	62
4.5.7	Model Creativity – Unconditioned Motion Generation	66
4.6	Conclusion	66
CHAPTER 5 DSTRUCT2DESIGN: DATA STRUCTURE DRIVEN GENERATIVE FLOOR PLAN DESIGN WITH TRANSFORMERS		
		68
5.1	Introduction	69
5.2	Related Work	70
5.2.1	Datasets	70
5.2.2	Generative Models	70
5.2.3	3D Scene Generation	71
5.3	DStruct2Design Floorplan Dataset	72
5.3.1	Data Structure	73
5.3.2	Datasets & Preparation	73
5.3.3	Bubble Diagrams	75
5.4	DStruct2Design (DS2D) LLM	75
5.4.1	Numerical Constraints	76
5.4.2	Bubble Diagrams	77
5.4.3	Complete Prompt	77
5.5	Metrics and Benchmarks	78
5.5.1	Self Consistency	78
5.5.2	Prompt Consistency	78
5.5.3	Summary	79
5.6	Experiments	80
5.6.1	Training Details	80

5.6.2	Hyper-parameters	81
5.6.3	Model Variants	81
5.7	Specific Attributes Sets in Preset Masking	83
5.7.1	Generation Prompts	83
5.8	Results	83
5.8.1	Creativity In Designs	87
5.9	Conclusion	87
CHAPTER 6 CONCLUSION		88
6.1	Cumulative Error from Iterative Process	88
6.2	Limit Number of Bounding Boxes	89
6.3	Video Generation Length	90
6.4	Latent Space Exploration	90
6.5	Text Visualization in Videos	90
6.6	Invalid Generations	91
6.7	Future Research	92
6.7.1	ViP-STEAM Follow Up	92
6.7.2	Ctrl-V Follow Up	93
6.7.3	DS2D Follow Up	93
REFERENCES		94
APPENDICES		108

LIST OF TABLES

Table 3.1	Summary of Dataset Statistics.	39
Table 3.2	Performance comparison with popular CNN-based models. The two variants of ViP-STEAM models used in the experiments are labeled in their names. We present results with the better of the two variants for each tasks – both variants are shown in the case of KTH-40 since they perform similarly. Their implementation details can be found in Section 3.3.1. The ViP-STEAM models marked with † are trained using the dataset configuration outlined in Table 3.1 and evaluated with the configuration specified in this table. We used padding to adjust any input dimension misalignments. ‡ The KTH-40 dataset employed in MCVD has a frame resolution of 64×64 , which is different from the resolution used in other studies. Prior work has evaluated methods using different combinations of past conditioning frames and future frames for prediction. We select past and future frame combinations which emphasize making long predictions in the future in such a way that we can compare with many prior methods providing results for these configurations.	40
Table 3.3	Ablation results for: feature-vs-frame recycling (FR) and STEAM blocks with-vs-without memory update (MU). The second column shows (FR) the status of the feature recycling, and the third column (MU) shows the status of the memory update.	43
Table 3.4	Analyzing the utility of STEAM blocks. We compare the performance of SimVP+gSTA model variants versus our ViP-STEAM-Trans approach on MMNIST10→50.	43
Table 3.5	A Loss Term Ablation Study. Columns one to three display the inclusion/exclusion status of various losses. To ablate the KL-divergence loss (Equation 3.8), we substitute the CVAE-Trans model with the STEAM-Trans model.	45
Table 4.1	Evaluation on generation quality. Generated videos are 25 frames with a resolution of 312×520	58

Table 4.2	All experiments reported in this table are conditioned on 3 initial bounding-box frames and 1 final bounding-box or trajectory frame. The first three columns show evaluations on the entire generated bounding-box sequence, measuring the alignment scores between our generated bounding box predictions and ground-truth labels. The last three columns focus on testing the auto-encoding capability of the network, evaluating only the first and last frames of the generated sequence. If a dataset’s name is labelled with “Traj” superscript, it indicates that in the experiment reported in that row, the last bounding-box frame is replaced with its trajectory frame.	60
Table 4.3	Motion Control Evaluation. We include Boximator [1] and trackDiffusion [2]’s reported performances on other datasets to provide a comparative context, though they use slightly different evaluation metrics. Additionally, in general, video length is inversely proportional to detection rate and generation quality. Our longer generation compared to related works poses an additional challenge for our model.	65
Table 5.1	Comparison to related works. Our method focuses on generation of floor plans as opposed to full 3D scenes, and it allows the user to specify individual room geometry (height, width, and/or area) and global apartment geometry in meters and sq. meters.	72
Table 5.2	Processed Floorplan Structure. Each row is a field in the structure. Brackets “[]” indicate the value of the field is a list. Indented rows below indicate fields in the list.	74
Table 5.3	Example of full instruction prompt used in training.	78
Table 5.4	Metrics and Associated Acronyms Used to Evaluate Generation Quality	79
Table 5.5	Summary of model variants	81
Table 5.6	Different presets of constraint set. Partial list represents information for only some of the rooms: dropout is applied on the individual room in the list of all rooms	83
Table 5.7	Our self- and prompt-consistency Benchmark results. Subscripts under model variants are generation prompts.	85
Table 5.8	Comparison of Compatibility↓ score (Graph Edit Distance) on the RPLAN dataset.	86
Table 5.9	Our self- and prompt-consistency on few-shot results	86
Table 5.10	Compatibility↓ score on ProcTHOR-trained models.	87
Table A.1	Glossary of terms.	109

Table A.2	Mappings of various dataset labels to MS COCO labels.	110
Table B.1	Self consistency metrics on generated results from Specific Prompt	120
Table B.2	Prompt consistency metrics on generated results from Specific Prompt .	120
Table B.3	Self consistency metrics on generated results from Total Area Prompt .	121
Table B.4	Prompt consistency metrics on generated results from Total Area Prompt	121
Table B.5	SC metrics on generated results from Partial Room Area Prompt	122
Table B.6	PC metrics on generated results from Partial Room Area Prompt	122
Table B.7	SC metrics on generated results from All Room Area Prompt . . .	123
Table B.8	PC metrics on generated results from All Room Area Prompt . . .	123
Table B.9	Compatibility ($\downarrow$) score on ProcTHOR experiments. The best score in each category (by number of rooms in a floor plan) is bolded, and the 2nd best score has an asterisk.	124
Table B.10	Self consistency metrics on RPLAN generations using Specific generation prompts.	125
Table B.11	Prompt consistency metrics on RPLAN generations using Specific generation prompts.	125
Table B.12	Self consistency metrics on RPLAN generations using Total Area generation prompts.	125
Table B.13	Prompt consistency metrics on RPLAN generations using Total Area generation prompts.	125
Table B.14	Self consistency metrics on RPLAN generations using Partial Room Area generation prompts.	126
Table B.15	Prompt consistency metrics on RPLAN generations using Partial Room Area generation prompts.	126
Table B.16	Self consistency metrics on RPLAN generations using All Room Area generation prompts.	127
Table B.17	Prompt consistency metrics on RPLAN generations using All Room Area generation prompts.	127

LIST OF FIGURES

Figure 2.1	Attention Module	13
Figure 2.2	Multi-headed Attention	15
Figure 2.3	Transformer architecture	16
Figure 2.4	U-Net architecture	22
Figure 2.5	Stable Diffusion model architecture	25
Figure 3.1	The ViP-STEAM CVAE-Trans Framework: the generation process within the ViP-STEAM CVAE-Trans network.	32
Figure 3.2	Architecture of the latent encoder and decoder.	33
Figure 3.3	Comparison of the gSTA-block [3] and the STEAM block	34
Figure 3.4	Iterative generation loops. The left Figure illustrates the original iterative process as described in SimVP [4, 3]. The right Figure depicts the modifications we have implemented in our approach.	37
Figure 3.5	Comparing generated results of SimVP versus our model on Moving-MNIST	41
Figure 3.6	Comparison between generated results of SimVP and our model on KTH-40	42
Figure 3.7	Examples of predicted results by ViP-STEAM.	42
Figure 3.8	Comparison between the last frame ($t = 30$) generated using feature recycling and frame recycling. In the feature-recycled generation (right), the Mercedes logo on the car is more distinct and the building windows are clearer compared to the frame-recycled generation (left).	45
Figure 4.1	The diagram illustrates two components of Ctrl-V : (left) the BBox Predictor and (right) Box2Video . For both models, we use a frozen, off-the-shelf VAE to encode images into latent space ($\mathcal{E}$) and decode them back into pixel space ($\mathcal{D}$). During training, (1) the BBox Predictor (Sec. 4.4.2) learns to denoise the noisy bounding box frame latents $\hat{\mathbf{b}}_t$, conditioned on the first ($\mathbf{b}^{(0)}$) and last ($\mathbf{b}^{(N-1)}$) bounding box frame latents and the padded initial frame latent $\mathbf{z}_{pad}^{(0)}$ and (2) the Box2Video (Sec. 4.4.3) denoises the target frame latents $\hat{\mathbf{z}}_t$ by conditioning on the initial frame’s latent $\mathbf{z}_{pad}^{(0)}$ (input to the SVD UNet) and the bounding box frame latents $\mathbf{b}$ (input to the ControlNet).	53
Figure 4.2	Comparison of an urban scene generation between the baseline Stable Video Diffusion (SVD) model and the teacher-forced Box2Video model.	59

Figure 4.3	Example visualization of a trajectory frame, where the bounding boxes are replaced by their mid-point traces.	60
Figure 4.4	Figure to explain mask metric definitions. Let the region of ground truth bounding box be $\mathbf{T}$ and predicted bounding box region be $\mathbf{P}$.	61
Figure 4.5	3D bounding-box predictions and motion-controlled video generations for various scenes on KITTI test-split.	63
Figure 4.6	Illustrations of the generations conditioned on ground truth 3D bounding boxes (2D for BDD) across various datasets. The 2D outlines of bounding boxes are overlayed on top.	64
Figure 4.7	Precision-recall curves: the x-axis represents the recall rate, and the y-axis represents the precision rate.	65
Figure 4.8	Bounding box trajectory generations are produced without conditioning on the last frame’s bounding box locations. The provided demos display generated trajectories with ground-truth bounding box trajectories overlaid for visual comparison.	66
Figure 4.9	2D bounding box frame generations and motion-controlled video generations on BDD100K test split where no final bounding box frame conditioning was provided.	67
Figure 5.1	Model Pipeline Overview	76
Figure 5.2	Example generations and extracted bubble diagrams. Pink background: ground truth floor plan and its associated bubble diagram. Green background: from F+BD model with Specific prompts. Blue background: from F+BD model with Total Area prompts.	84
Figure 6.1	Comparison of One-Way sign between ground truth and generated video frame.	91
Figure 6.2	Text in the driving lane morphs over time.	91
Figure 6.3	Floorplan generated with the same specification.	92
Figure A.1	YOLOv8 detections on BDD and vKITTI with confidence thresholds set to 0.1. The detection confidence scores are labeled on the detected bounding boxes.	110
Figure A.2	2D bounding-box frame predictions and motion-controlled video generations for various scenes.	112
Figure A.3	2D bounding-box frame predictions and motion-controlled video generations for various scenes.	113
Figure A.4	3D bounding-box predictions and motion-controlled video generations for various scenes on vKITTI test-split.	115

Figure A.5	3D bounding-box predictions and motion-controlled video generations for various scenes on vKITTI train-split.	116
Figure A.6	3D bounding-box predictions and motion-controlled video generations for various scenes on KITTI test-split.	118

LIST OF SYMBOLS AND ACRONYMS

CNN	Convolutional Neural Network
CVAE	Conditional Variational Auto-Encoder
DDPM	Denoising Diffusion Probabilistic Model
DM	Diffusion Model
FVD	Frechet Video Distance
GAN	Generative Adversarial Network
GRU	Gated Recurrent Network
GPT	Generative Pre-Trained Transformers
KL	Kullback–Leibler
LLAMA	Large Language Model Meta AI
LLM	Large Language Model
LDM	Latent Diffusion Model
LPIPS	Learned Perceptual Image Patch Similarity
LSTM	Long Short-Term Memory Network
LoRA	Low-Rank Adaptation
NN	Neural Network
PSNR	Peak Signal-to-Noise Ratio
RNN	Recurrent Neural Network
SSIM	Structural Similarity Index Measure
SVD	Stable Video Diffusion
VAE	Variational Auto-Encoder

LIST OF APPENDICES

Appendix A	Ctrl-V: Additional Results	108
Appendix B	DS2D: Additional Results	119

CHAPTER 1 INTRODUCTION

1.1 Motivation

What is artificially intelligence? In Alan Turing's famous work "Computing Machinery and Intelligence" [5] in 1950, he outlined the possibility of having machines that think, and devised the famous Turing Test: a machine is said to be "thinking" if it can hold a conversation while providing responses that are indistinguishable from those given by a human. And in 1956, at the Dartmouth workshop, the phrase, "Artificial Intelligence", was coined and it became a formal academic discipline. The workshop had a grand goal of creating machines capable of simulating human intelligence, it is said to investigate the hypothesis that "every aspect of learning or any other feature of intelligence can be so precisely described that a machine can be made to simulate it" [6].

However, these early definitions of intelligence have evolved over time. For instance, IBM's Deep Blue, which excelled at playing chess through rule-based expert systems, was once considered intelligent but is now viewed as less sophisticated compared to newer approaches like AlphaZero [7]. The concept of intelligence has transformed across decades, from solving mazes with heuristic search in the 1960s to creating rule-based expert systems in the 1980s and using probabilistic models and intelligent agents in the 1990s and 2000s.

Today, I believe intelligence to be the ability to express thought. From Joseph Weizenbaum's ELIZA [8] to OpenAI's GPT [9], this is a type of intelligence that researchers have strived to achieve from the very beginning, a definition of intelligence that withstood the test of time. The intrinsic nature of expressions based on a thought or an emotion is closely aligned to perspectives taken by generative models researches – which will be explored in the following sections. This thesis is motivated by the idea that intelligence is not solely about solving specific problems or tasks but rather about the ability to generate and express thoughts, a capability that is increasingly important in AI systems.

1.2 Intelligence Through Expression

The ability to extract a thought and express it is a landmark of intelligence. Evident in various aspects of our daily lives, this trait enables us to convey our ideas, emotions, and experiences through different mediums, such as art, literature, music, and even everyday conversations.

This is exemplified in many ways in the realm of art, literature and music. Painters pour their thoughts on canvas, creating art pieces based on their perception of reality. Sculptors shape their thoughts by hands into tangible forms, creating sculptures based on their vision. Photographers frame their thoughts through lenses, capturing moments of beauty based on their perspective. Writers distill their thoughts and emotions into words, crafting stories that evoke more thoughts and emotions from the readers. Musicians compose melodies and lyrics that reflect their thoughts and experience, resonating emotions in those who listen.

Beside these creative expressions, this process is also pervasive in our daily interactions. It happens when we engage in conversations, where we extract our opinions to share with others, forming them into spoken language to convey our thoughts. It happens on social media when we send a post, where we condense our thoughts, forming them into written language, sharing our experiences and perspectives with the world. It happens at work when we create a design that fits specific criteria, where we distill a product from a web of thoughts understanding the constraints. Even in seemingly mundane tasks like deciding what to have for dinner, or where to visit for the weekend, we extract our preferences and priorities then express them as choices that reflect our thoughts and desires.

On a higher level, the impact of this ability is felt far beyond the realm of creative expression. In the scientific community, researchers and scientists extract insights from data, expressing them as theories and hypotheses that shape our understanding of the world. In the business world, entrepreneurs and innovators extract ideas from market trends, expressing them as products and services that transform industries. Even in our personal relationships, we extract emotions and thoughts, expressing them as empathy and understanding that foster deeper connections.

Indeed, the significance of this ability extends beyond individuals. It enables us to build upon each other's thoughts and ideas, in science, in art, in our everyday lives, driving collective progress and innovation. Through the expression of thought, we create a growing shared knowledge base upon which advancement is made. In the digital age, this process has been amplified, with the internet and social media providing platforms for thoughts to be extracted and expressed on a global scale.

As we continue to develop artificial intelligence, it is essential to recognize the importance of expression and strive to replicate it in machines. By creating AI systems that can extract and express thought, we can unlock new possibilities for human-AI collaboration, creativity, and problem-solving.

Fortunately, current research in AI heavily focuses on the field of generative models, which intrinsically mirror the expression process. As such, this thesis explores the field of generative

models to contribute to the process of achieving intelligence in expression. Specifically, this thesis presents three work on generative models: two on video generation and one on text generation. Among these work, this thesis introduce novelty controlled generation paradigm that provides more fine-grained specifications to the expression generation process. In the next sections, we first introduce generative models and controlled generation to motivate the direction of this thesis, then gives an outline of the major chapters to follow.

1.3 Generative Models

Generative models are a class of unsupervised machine learning models that aim to generate new data samples, given a set of data $\mathcal{X}$. This data can be paintings, driving videos, story books, speeches etc. The assumption is that there is an underlying distribution p_{data} from which this set of data originate from. And the the goal is to learn a probability distribution $p_{generator}$ over $\mathcal{X}$ such that $p_{generator}$ approximates p_{data} to allow for the generation of new, synthetic data samples that resemble the real data.

As such, they are the ideal models to learn various expressions, be it art, literature, or video content. The rapid development in the field of generative models have introduced various generative models for different types of data.

The most prominent generative model is perhaps Generative Adversarial Networks (GANs), introduced in 2014 by Goodfellow et al. [10], provides novel two-component adversarial approach to learning the data distribution. As a brief overview, a GAN contains a Generator (G) and a Discriminator (D). The Generator is a neural network that takes a random noise vector as input and produces a synthetic data sample that attempts to mimic the real data. The Discriminator is also a neural network, and it takes a data sample (real or synthetic) as input and outputs a probability that the sample is real. During training, the Generator tries to produce realistic data to fool the Discriminator, while the Discriminator tries to correctly distinguish between real and fake data. This adversarial process leads to both improvement in both networks, and it ultimately results in a robust Generator that can produce highly realistic data samples. However, in recent works, diffusion models have overtaken GANs as the dominant method for generative models of images and videos.

Diffusion models, specifically, Denoising Diffusion Probabilistic Models (DDPMs), are a main focus of our video generation work (Ctrl-V) in Chapter 4. DDPM works by iteratively adding Gaussian noise to the data and learning a reverse process to restore the noisy data into its original form. In doing so, the model learns the underlying data distribution and can create new data sample from noise. A more detailed explanation of diffusion models are in Section

2.5.

Most video diffusion approaches operate in some kind of latent space of videos. Input video data are often processed by a Variational Auto-Encoder (VAE) encoder first, which transforms the video data into a latent representation, before undergoing the diffusion process. More specifically, an VAE [11] is a type of generative model that combines the capabilities of autoencoders and generative models. It consists of two main components: an encoder and a decoder. The encoder is a neural network that maps the input data to a lower-dimensional latent space, typically denoted as z . The decoder is a neural network that maps the latent space back to the original data space, generating a reconstructed version of the input data.

In our video generation work (ViP-LVM) presented in Chapter 3, we incorporate a Variational Auto-Encoder (VAE) in one of our model variants, as an attempt to see how far we can push light weight CNN-based video generative model with a VAE style latent transformation of video data. And in our Ctrl-V work presented in Chapter 4, we also employ VAE encoder and decoder. Video data always pass through a widely used pretrained VAE image encoder in order for our diffusion based model to process it in latent space, and the output of the our model is reconstructed into image frames through the corresponding VAE decoder.

The last major generative model used in our work is the class of transformer-based language model. In our work in text generation (DStruct2Design) in Chapter 5, we fine-tune a Large Language Model (LLM). This type of model consists of only transformer decoders (Section 2.2.3). The autoregressive decoding process enables the model to generate text one token at a time, predicting the next token based on the context provided by the previous tokens. This process is driven by the language modeling objective, which trains the model to predict the next token in a sequence, given the context of the previous tokens. A detailed explanation of transformer-decoder generative model is presented in Section 2.2.3.

1.4 Controlled Generations

Controllable generation in generative models refers to the ability to generate content that meets specific criteria, or conditioned on specific inputs. This is achieved by incorporating additional inputs, constraints, or conditioning factors into the generative model, allowing it to focus on specific aspects of the output. So instead of learning a data distribution over the data set, $p_{generator}(\mathcal{X})$, it learns a conditional distribution $p_{generator}(\mathcal{X}|\mathcal{Z})$ where $\mathcal{Z}$ is the set of control criteria that the inputs are conditioned on. In all three work presented in this thesis, we employ a conditional generation paradigm.

In Chapter 3, our video generation approach is formulated as generating frames in a blockwise

manner. The generated blocks of frame is always conditioned on the previously generated block of frames and a distilled history of all past frame blocks. In addition, in one variant of our conditional generation model, we also incorporate a VAE which in itself has an encoder that we modify to take the conditioning as part of the input.

In Chapter 4, our video generation takes bounding boxes as conditioning to dictate what and where objects should appear in the videos. We introduce several novel mechanisms to incorporate this control input in our two-step model process.

And In Chapter 5, our text generation is explicitly conditioned on a set of user constraints in a special text form that outlines a specific data-structure.

1.5 Outline

This thesis takes a classic thesis approach as the work presented in this thesis are mostly under review. Even though this is not a thesis by article, arXiv versions of the contributions are available. The rest of this section outlines the structure of the remaining chapters of this thesis.

Chapter 2: Background and Literature Review

In this chapter, we review relevant literature on generative models for text and then for videos. Instead of a critical review of specific work that relate to each of the three work presented in this thesis, we opt to focus on background work that build up to the most important models in text and video generation. We defer specific literature reviews on each of the three work to to their own chapters in the style that is used for academic papers.

We start with definition of an auto-regressive model in Section 2.1 before moving on to briefly reviewing recurrent neural networks (RNN) and long short-term memory networks (LSTMs). We then give a detailed overview of transformers in Section 2.2 and their core attention mechanism. We then give a list of popular widely used models for generative text models, based on the transformer encoder or decoder blocks. We also present efficient fine-tuning algorithms that are used in our work.

We briefly review convolutional neural network (CNN) in Section 2.3 as the start of the body of works relevant to image and video generative models. We provide a brief explanation on how CNNs work along with the mathematics behind it, before moving on to important vision models that are built with CNNs. In particular, we present the inner workings of U-Net and 3D U-Net Section 2.4 as they are used in our Ctrl-V work (Chapter 4). Last but not least,

we review diffusion models in Section 2.5 as they are the backbone models for our Ctrl-V work.

Chapter 3: Video Prediction via Spatio Temporal External Attention and Memory

In this work we focus on the light-weight CNN-based video prediction models which are often overlooked due to the explosion of heavy-weight diffusion based models.

We identify three pressing problems with current state of the art CNN-based model SimVP. Firstly, we note that the model can only take in limited context due to its block-to-block prediction structure, causing it to effectively only condition on a single block of frames in generating new frames. This means the model does not retain context from frames prior to the block of frames used. Secondly, we note that the model’s generation suffers from significant image degradation due to the inner model structure which involves repeated encoding and decoding of images. This causes small errors to compound and explode over the prediction iterations. Lastly, we note that the model lacks stochasticity which leads to blurry predictions in situations with multiple possible outcomes. In the chapter, we propose solutions to all three issues in our novel ViP-STEAM model.

We start by breaking down the model structure in Section 3.3: we explain the functions of the encoder, decoder, and translator. We then dive into details of the novel STEAM blocks, which are the core of our model. In that section, we outline the process of how STEAM blocks perform memory update with the added memory mechanism. This is a crucial step we introduce to combat the problem of limit context that exists in SimVP.

In the same section, we introduce our STEAM-Trans model variant which uses a conditional VAE (CVAE) in the translator in order to capture stochasticity. This is a solution to the lack of stochasticity present in the SimVP model

In Section 3.3.2, we present our solution of feature recycling to mitigate the issue of image degradation. We show how we shift the task of frame translation (block prediction) from the image space to the latent space, thereby avoiding repeated encoding and decoding of image frames which introduce cumulative errors into the final video.

We provide more training details in Section 3.3.3, including the various training objective used to ensure all different components of the model works as we expect.

We outline the specifics of the experiments in Section 3.4, including what datasets we used, how pretraining is done, and how the experiments are configured. We provide an analysis on the results in Section 3.4.4 and show that our model achieves state of the art or competitive

results on all benchmark datasets used.

To investigate the effectiveness of our novel components, we run an detailed ablation study and present its results in Section 3.5. We show that the STEAM-Trans model variant shows insignificant change to capturing the stochasticity of videos, but overall, our introduction of STEAMblocks, memory updates, and feature recycling all help improve the performance of the model.

This work is in the process of being submitted to the IEEE/CVF Winter Conference on Applications of Computer Vision 2025, before July 15 2024, and we plan to submit it onto arxiv as well. Much of the work presented here will overlap with the arxiv version.

Chapter 4: Ctrl-V: Higher Fidelity Video Generation with Bounding-Box Controlled Object Motion

In this work, we move from light weight CNN-based model to diffusion models to explore controlled video generation. We show that current approaches to controlled video generation mostly involves complex modes of control such as depth map, optical flow map etc, which are costly to adjust and edit. Additionally, most methods introduce special adaptive layers that are specific to the mode of control in interest. In comparison, we make use of bounding boxes as a flexible control mode in that it is cheap to edit and adjust, and we render them in pixel space to avoid using specialized adapters to understand the control input. We create a flexible way to use this control signal and generate real world or simulated driving videos that have objects consistent with bounding box placements in every single frame. Specifically, we introduce our Ctrl-V model which consists of a bounding box predictor network, BBox Predictor, that render pixel-level bounding boxes, and a video predictor, Box2Video, that uses these bounding boxes as conditioning to create high-quality video.

We first review relevant methods in Section 4.3. This include different current approaches to conditional or controlled video generation. We move on to methods that specifically uses bounding boxes as their control mode and highlight the difference between our method and each of those approaches.

We present our method in Section 4.4. This section is divided into two part, one for each component of our two-part model. We first start with the BBox Predictor model. We explain how bounding boxes are preprocessed into informative image renders before moving on to the structure and the core U-Net component within the BBox Predictor in Section 4.4.2. We focus on the Box2Video component, which is responsible for combining the control input into the video generatino process and creating a final video that is consistent with the input

bounding boxes, in Section 4.4.3. In it, we show how we adapt a ControlNet on top of the Stable Video Diffusion pipeline in order to assert control on the generated video.

We write on the specifics of the experiments, the datasets involved, and the metrics used to evaluate our results in Section 4.5. Notably, most existing paper uses their own variant of the mean average precision metric in their respective work, and they are not exactly suitable for the problem we tackle. We explain how we implement our mean average precision uses it in our evaluation.

We present our evaluation results on the video generation quality in Section 4.5.4 along with image demonstration from the generated videos. We also ablate and evaluate our two model components separately and presents the results on BBox Predictor and Box2Video in Section 4.5.5 and 4.5.6 respectively.

Overall we demonstrate the effectiveness of the BBox Predictor model in generating bounding box sequences and the quality of Box2Video’s video generations, as well as how strictly they adhere to the control bounding boxes.

This work has been submitted to Conference on Robot Learning 2024. An arxiv version of the paper has been submitted to arxiv as well. Much of the content in the arxiv version overlaps with what is presented here.

Chapter 5: DStruct2Design: Data Structure Driven Generative Floor Plan Design with Transformers

In this work, we move from controlled video generation to controlled text generation. Specifically, we formulate the floorplan design problem, which traditionally is treated as a image generation task, as a text generation challenge. In doing so, we enable the use of numerical constraint as conditioning when generating a text representation of a floorplan. This more closely mirrors the real world problem of floorplan design where the client usually have quantitative requirements like square footage, and sizes of individual rooms etc. In addition, we also design our text representation to be compatible with other work, in the sense that the usual input of "bubble diagram" (a design graph that denotes the connectivity between rooms) are also processed by our model and can be used as part of the control signal. Moreover, our text representation is easily transformed into images to be compatible with image-based formulation of the problem.

In Section 5.3, we present the data structure we use to represent floorplan images. We explain that it contains numerical information about the floorplans in order to be readily evaluated against the input constraints used as conditions, and it is represented in text directly as a

string of the data structure in order for the model to draw parallels against the conditions in the input which are also in the same data structure form. We go into more details about the datasets used and how the data are converted.

We give an overview on bubble diagrams in Section 5.3.3, before moving onto details on our DStruct2Design (DS2D) LLM is set up and fine-tuned in Section 5.4. In that section, we include specifics about constructing the numerical constraints and bubble diagrams into texts, and how they are combined with instructions to form a control prompt in the training process.

To accomodate this new formulation, we develop a new suite of metrics that helps evaluate how consistent the numerical values in the generated floorplans are, both within the floorplan and with respect to the control inputs. The description of each individual metric is listed in Section 5.5.

Multiple model variants are required to accomodate different control strategies, and we list them out in Section 5.6. Selected model evaluation and results are shown in Section 5.8, and the entire evaluation on all of the model variants are listed in Appendix B

Overall, we show that text-based generation results in floorplans that follow the numerical constraints suprisingly well. It also generate floorplans that abides by the bubble diagram input to a degree competitive with the state of the art image based models.

This work has been submitted to the thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024). We plan to submit it onto arxiv as well. Much of the work presented here will overlap with the arxiv version.

Chapter 6: Conclusioin

In the last chapter of the thesis, we summarize the importance of the works presented in the thesis. We discuss some of the current challenges discovered during the works and how they can be improved upon. We conclude the thesis with what future work these works enable.

CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

In this Chapter, we provide background on the concepts and models that build up to the most important recent models in text and video generative models. Despite this thesis taking a classic thesis structure, we opt to put important specific literature reviews within each chapter in the style that is used for academic papers.

2.1 Early Autoregressive Models

We start by providing background knowledge on the development of text-based generative models. These models are recurrent and autoregressive in nature. One of the most widely-used and perhaps the earliest form of autoregressive neural network model is the Recurrent Neural Networks (RNNs). The emergence of RNN follows neural network work by Hopfield [12] and back propagation innovations by Rumelhart et al.[13]. Along with Long Short-Term Memory networks (LSTMs) [14], which are recurrent networks with specialized gates, this family of autoregressive model opened the door to early ideas of conditional generation where the output at time step t is dependent on results from time step $t - 1$. It inspires a series of recurrent-based models for conditional video generation such as predRNN [15], convLSTM [16], E3D [17] etc. which we will discuss later on. Here, we provide a brief review on RNN and LSTM in the following sections.

2.1.1 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural networks designed for processing sequential data. Unlike feedforward neural networks, RNNs have connections that form directed cycles, allowing information to persist. This persistence makes RNNs particularly suitable for tasks where context from previous inputs is necessary, such as time series prediction, language modeling, or even video generation.

RNN Equations

Given an input sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$, the RNN computes the hidden states $\mathbf{h} = (\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T)$ using the following equations:

1. Hidden state update:

$$\mathbf{h}_t = \sigma(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \quad (2.1)$$

where:

- $\mathbf{h}_t$ is the hidden state at time t
- $\mathbf{x}_t$ is the input at time t
- $\mathbf{W}_{hx}$ is the weight matrix for the input
- $\mathbf{W}_{hh}$ is the weight matrix for the hidden state
- $\mathbf{b}_h$ is the bias vector
- σ is the activation function, typically tanh or ReLU

2. Output computation:

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y \quad (2.2)$$

where:

- $\mathbf{y}_t$ is the output at time t
- $\mathbf{W}_{hy}$ is the weight matrix for the output
- $\mathbf{b}_y$ is the bias vector

2.1.2 Long Short-Term Memory Networks (LSTMs)

Long Short-Term Memory (LSTM) networks are a type of RNN specifically designed to address the problem of long-term dependencies, which traditional RNNs struggle with due to issues like vanishing and exploding gradients. LSTMs introduce a more complex architecture with additional components called gates, which regulate the flow of information.

LSTM Equations

LSTMs introduce three gates (input, forget, and output gates) and a memory cell c_t . Given an input sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$, the LSTM computes the hidden states $\mathbf{h} = (\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T)$ and cell states $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_T)$ using the following equations:

1. Forget gate:

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f) \quad (2.3)$$

2. Input gate:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{b}_i) \quad (2.4)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c) \quad (2.5)$$

3. Cell state update:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (2.6)$$

4. Output gate:

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o) \quad (2.7)$$

5. Hidden state update:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (2.8)$$

where:

- σ is the sigmoid function
- $\odot$ denotes element-wise multiplication
- $\mathbf{W}_{fx}, \mathbf{W}_{fh}, \mathbf{W}_{ix}, \mathbf{W}_{ih}, \mathbf{W}_{cx}, \mathbf{W}_{ch}, \mathbf{W}_{ox}, \mathbf{W}_{oh}$ are weight matrices
- $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_c, \mathbf{b}_o$ are bias vectors

The LSTM structure allows the network to decide when to forget previous hidden states and when to update them, making it more effective for learning long-term dependencies compared to traditional RNNs.

2.2 Transformers

In 2017, the Transformer architecture made its debut in the work "Attention is All You Need" by Vaswani et al.[18] It has since become one of the foundation building blocks for machine learning research, especially in the area of language modeling. To understand transformers, we first need to understand its main novelty: attention mechanism.

2.2.1 Attention

Earlier recurrent structured networks are well known to struggle when it comes to capturing long term dependency in sequences. Even though later modifications, such as those proposed in the LSTM or GRU architectures, tries to improve on this aspect, they still fall short by today's standard. This is partly due to the fact that these structures maintain a single internal state, and when the state gets repeatedly updated at each time step, some relevant early information is unavoidably lost.

Attention mechanism, proposed as part of the transformer architecture, combats this known limitation and allows the model to weigh the importance of different words (in context of language modeling) in a sentence regardless of their distance from each other. Specifically, rather than maintaining a single state vector like prior recurrent methods, it creates separate representations for input at each time step. In doing so, it allows the input at any time step to query input at any other time step, regardless of how far apart they are.

Self Attention

Mathematically, given an input sequence $\mathbf{X}$ with length n and dimension d :

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$$

where $\mathbf{x}_i \in \mathbb{R}^d$.

The attention mechanism first creates 3 representations for each x_i : Query, Key, and Value representations:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V \quad (2.9)$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_k}$ are learned projection matrices, and $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d_k}$.

To compute the final representation of each input x_i , informed by all other x_j 's, we take their respective query vector, and perform dot product against all other x_j 's key vectors. This product is normalized by a factor of $\sqrt{d_k}$ and through a softmax function, and the normalized results are used as weight: it tells us how much attention the current x_i should pay to each x_j . To arrive at the final representation for x_i , we add up the value vectors of

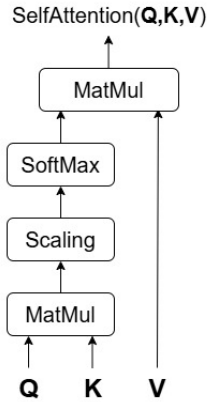


Figure 2.1 Attention Module

the x_j 's, weighed by the weight factors. This can be succinctly summarized in matrix form:

$$\text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V} \quad (2.10)$$

where $\mathbf{Q}\mathbf{K}^T \in \mathbb{R}^{n \times n}$ represents the dot-product of queries and keys.

Cross Attention

In addition to self attention, transformer also employs cross attention. Cross attention arises due to the encoder-decoder nature of the transformer architecture. On the decoder side, the output at each time step also perform attention over the encoder inputs – it wants to know what part of the input it needs to pay attention to. Therefore, the query vector for that decoder output is run against the key vectors of the encoder inputs, and a weighted sum of the encoder values are used as part of the output representation. Mathematically, cross attention is largely the same with self attention, it only substitutes the $\mathbf{x}_i$'s by $\mathbf{y}_i$'s.

In matrix form this looks like:

$$\mathbf{Q} = \mathbf{Y}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V \quad (2.11)$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_k}$ are learned projection matrices, and $\mathbf{Q} \in \mathbb{R}^{m \times d_k}$, $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d_k}$.

$$\text{CrossAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V} \quad (2.12)$$

where $\mathbf{Q}\mathbf{K}^T \in \mathbb{R}^{m \times n}$ represents the dot-product of queries from the target and keys from the source.

Multi-Headed Attention

Multi-Headed Attention extends the self-attention mechanism by running multiple attention mechanisms, or heads, in parallel. As shown in Figure 2.2, each head has a different set of learned projections, and their outputs are combined at the end. In doing so, the model can attend to different parts, each focusing on a different aspect of language.

Mathematically, each head i (where $i \in \{1, \dots, h\}$ and h is the number of heads) has its own query matrix $\mathbf{Q}_i$, keys matrix $\mathbf{K}_i$, and values matrix $\mathbf{V}_i$, and they project input $\mathbf{X}$ just like

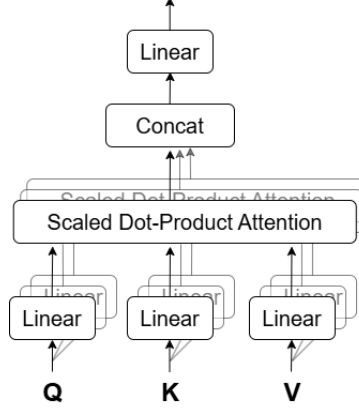


Figure 2.2 Multi-headed Attention

individual attention mechanisms reviewed earlier:

$$\mathbf{Q}_i = \mathbf{X}\mathbf{W}_{\mathbf{Q}_i}, \quad \mathbf{K}_i = \mathbf{X}\mathbf{W}_{\mathbf{K}_i}, \quad \mathbf{V}_i = \mathbf{X}\mathbf{W}_{\mathbf{V}_i} \quad (2.13)$$

$$\text{head}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) = \text{softmax} \left(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{d_k}} \right) \mathbf{V}_i \quad (2.14)$$

However, in multi-head attention, the final outputs from all h heads are concatenated and transformed by a learned matrix to arrive at the final representation:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h) \mathbf{W}_O \quad (2.15)$$

$\text{Concat}(\cdot)$ denotes concatenation along the feature dimension, resulting in a matrix of size $n \times (h \cdot d_k)$, and $\mathbf{W}_O \in \mathbb{R}^{(h \cdot d_k) \times d}$ is a learned weight matrix.

2.2.2 Encoder and Decoder Architecture

The original paper by Vaswani et al. [18] presents the an encoder-decoder model. The encoder is composed of a stack of N identical encoder blocks, with each block containing a self-attention mechanism to attend over the input sequence, and a feed forward layer to transform input at each position into suitable representations. The decoder block is similarly built, with the main difference being that it perform "masked" attention instead of self-attention. This means for every target word y_t at position t , the decoder only attends to words up to t . Furthermore, each decoder block has an additional layer to perform cross attention in order to determine how each input word should affect the current output word.

The transformer encoder-decoder model are further explored in popular work such as T-5

[19]. However, over time, transformers have been adapted and evolved beyond this original structure to address the diverse needs and challenges of different tasks in natural language processing (NLP). Transformer-based models mostly use either only the encoder or the decoder from the original paper.

Encoder-only transformers, such as Bidirectional Encoder Representations from Transformers (BERT) [20], are designed for tasks requiring understanding and contextualization of a single text sequence, without the need to create new context. It excels in tasks such as text classification, question answering and named entity recognition. These tasks do not inherently require sequence generation, making the decoder component unnecessary. The encoder in BERT is trained to produce contextualized embeddings by using masked language modeling (MLM) and next sentence prediction (NSP).

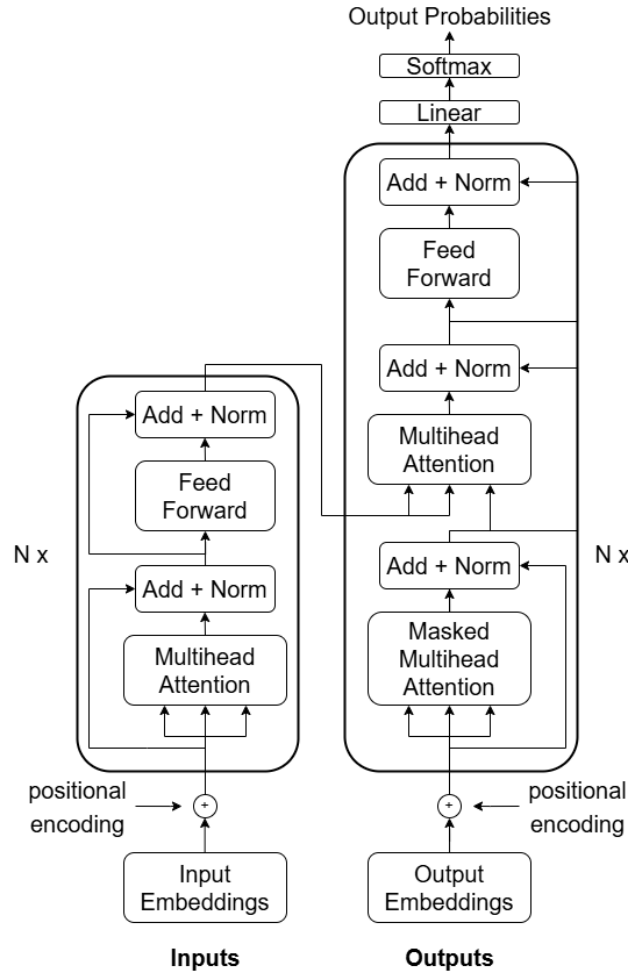


Figure 2.3 Transformer architecture

Decoder-only transformers, such as Large Language Model Meta AI (LLaMA) [21] Gener-

ative Pre-trained Transformer (GPT) [9], focus on tasks that involve text generation, like language modeling. These models are designed to predict the next token in a sequence given the preceding tokens, operating in an autoregressive manner. The simplicity of using only the decoder allows for efficient generation and scaling. As such, they enable well-defined conditional generation, and are the focus of one of our chapters on design generation with data structure as text. We will proceed to expand on decoder-only transformers in the next section.

2.2.3 Decoder-only Transformers

Decoder-only transformers excel in autoregressive text generation tasks. In these models, text is generated one token at a time. Each token is predicted based on the sequence of previously generated tokens. Along with self-attention, this sequential approach allows the model to produce coherent and contextually relevant text. Language models that uses decoder-only transformers, such as GPT [9], LLAMA [21], are becoming more and more prevalent.

This autoregressive process begins with an initial input sequence. The model first tokenize the sequence, and map each token into a vector according to a dictionary of embeddings. Token position information is then added via the use of positional embeddings. These positional embeddings can vary from absolute position embedding like those used in the original transformer paper, to Rotary Position Embedding [22] as those used in LLaMA-2 [23], or to even newer position-less positional embedding like ALiBi [24]. The final token embeddings then go through self attention to arrive at their state representation, which the model uses to generate a probability distribution over the potential next tokens. The model selects the next token by sampling from this distribution and appending it to the input sequence. This updated sequence is then fed back into the model to predict the following token. This iterative generation process continues until a specified stopping criteria is met – such as reaching a predefined sequence length or when a end-of-sentence token is generated.

This approach leads to several natural ways to control generations.

2.2.4 Controllable Text Generation

Controllable text generation is where the output can be influenced or conditioned by specific inputs or constraints. This capability is achieved through several conditioning methods:

Prompting

Prompting is the simplest method. It involves designing the initial input sequence, or prompt, to guide the model's output. For example, a prompt like "Write a story about a musician" directs the model to generate a story on the topic of musicians. This approach relies on the model's understanding of the context provided by the prompt, and it depends on the data used in training the language model.

Prefix Conditioning

This approach involves adding a control sequence to the beginning of the input. For example, prepending "[Topic: Science]" or task specific sequences like "[Translate to English]" to the input sequence biases the model towards generating text related to control sequence. This method effectively sets a context that influences the subsequent text generation, and models like CTRL [25] employing this approach shows coherent and on topic text generations.

Prefix-Tuning

Instead of fine-tuning the whole network, work by Li et al. [26] propose to fine tune special tokens that are put in front of the input sequence. This is similar to Prefix Conditioning, but only the special tokens are trained.

Model Fine-Tuning

Fine-tuning language model is the main method we can concerned with in this thesis. It involves taking a pretrained language model, and adapting its general language understanding capabilities to a specific task or domain by training it on a task-specific dataset. These data often come in pairs, such as summaries and full articles. It also may contain special prefixes or follow specific format for the model to learn. During the fine-tuning process, the model's weights are adjusted to minimize the loss function specific to the task. During this phase, the model learns to apply its pretrained knowledge to the new context, effectively transferring its generalized understanding to perform well on the specialized task. After training, the model is evaluated on a validation set to ensure that the model generalizes well.

2.2.5 Efficient Fine-Tuning

As larger and larger language model emerges, the time and monetary cost of fine-tuning these models increase. Finding efficient and low-cost method to fine tune large language models

are crucial. There are two widely-used methods to aid the process of fine-tuning LLMs: Low-Rank Adaptation (LoRA) [27] and Low Bit Quantization [28]. For this thesis, the Low Bit Quantization algorithms used are specifically AWQ [29] and GPTQ [30]

Low-Rank Adaptation

Low-Rank Adaptation (LoRA) is a technique designed to fine-tune pretrained models more efficiently. Instead of updating all the model’s parameters, LoRA introduces low-rank decomposition into the parameter matrices. This approach significantly reduces the number of trainable parameters by representing the weight updates as a product of two smaller matrices.

Given a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$, where d and k are the dimensions of the input and output features, respectively, LoRA approximates the weight update $\Delta\mathbf{W}$ as a product of two smaller matrices:

$$\Delta\mathbf{W} \approx \mathbf{A}\mathbf{B} \quad (2.16)$$

Here, $\mathbf{A} \in \mathbb{R}^{d \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times k}$ are low-rank matrices, with $r \ll d, k$. The rank r is a hyperparameter that controls the trade-off between accuracy and number of parameters.

During fine-tuning, instead of updating the full weight matrix $\mathbf{W}$, only the smaller matrices $\mathbf{A}$ and $\mathbf{B}$ are learned. During testing, the updated weight matrix can be expressed as:

$$\mathbf{W}' = \mathbf{W} + \Delta\mathbf{W} = \mathbf{W} + \mathbf{A}\mathbf{B} \quad (2.17)$$

The key advantage of LoRA is its computational efficiency. By using low-rank matrices $\mathbf{A}$ and $\mathbf{B}$, the number of trainable parameters is reduced from $d \times k$ to $r \times (d + k)$, which is significantly smaller when $r \ll d, k$. This reduction in trainable parameters leads to faster training times and lower memory consumption.

Bit Quantization

Bit quantization is a method used to reduce the memory footprint and computational demands of models such as large language models. This technique essentially represents the model’s weights and activations with fewer number of bits than the standard 32-bit floating-point precision. Like LoRA, this method also trade efficiency for accuracy, the lower the number of bits, the more information loss. Common quantization levels include 8-bit, 4-bit, or even lower precision. By reducing the bit-width, quantization decreases the model’s size

and accelerates inference times, making it feasible to fine-tune large models with less time and hardware resources. However, quantization can introduce some loss in precision, potentially affecting model performance.

2.3 Convolutional Neural Network

In the following sections, we provide relevant background knowledge on image and video generative models.

Convolutional Neural Networks (CNNs) [31, 32] are a class of deep neural networks specifically designed for processing data with a grid-like topology, such as images. Unlike traditional neural networks, which treat each pixel of an image as an independent input, CNNs leverage the spatial structure of the data. This makes them particularly effective for tasks such as image classification, object detection, and segmentation. They have become the building block of more complex image and video processing models such as Unet [33], which we will discuss in the next section. In this section, we outline the basics of CNNs.

CNNs are mainly made up of convolutional layers. Each convolutional layer consists of a set of filters (also known as kernels) that are convolved with the input image. Each filter is a patch that slides across the input image and performs an element-wise multiplication and summation. This summation results in a feature map. These filters are capable of detecting various features such as edges, textures, and patterns. The convolution operation reduces the spatial dimensions of the input, allowing the network to learn hierarchical patterns efficiently.

Mathematically, the convolution operation for a single filter can be expressed as:

$$h_{i,j} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x_{i+m,j+n} \cdot \mathbf{W}_{m,n} + b \quad (2.18)$$

where:

- $h_{i,j}$ is the output value at position (i, j) in the feature map.
- x is the input image or the input from the previous layer.
- w is the filter (or kernel) with dimensions $M \times N$.
- b is the bias term.
- The summation is over the spatial dimensions of the filter.

Following the convolutional layers, activation functions such as the Rectified Linear Unit (ReLU) [34] are applied to introduce non-linearity into the model, which is essential for learning complex patterns. Following the activation function, pooling layers are typically used to further reduce the dimensions of the feature maps. Max pooling is the common operation of choice, it selects the maximum value from a patch of the feature map, which effectively performs down-sampling while retain the most important information.

As the image passes through multiple convolutional and pooling layers, the network captures increasingly abstract features. These high-level features are then fed into fully connected layers, where each neuron is connected to every neuron in the previous layer. The output of fully connected layers can then be used for various tasks such as classification. Even without the fully connected final output layer, the output of the last convolutional layer can still be used as a abstract representation of the image. This comprehensive architecture allows CNNs to learn both low-level and high-level features, making them highly effective for any task involving images.

Many popular image processing networks such as the Inception Network series [35], or the VGG network variants [36] are all built on CNNs. However CNN-based networks quickly run into a problem of having too much depth and features from early layers being lost. The introduction of skip connection, like those used in U-Net mitigates this problem. Nowadays, U-Net [33] has become the go-to structure for processing images in current research. In one of our work on video generation, we also make use of U-Net. In the following section, we expand on what U-Net is.

2.4 U-Nets

U-Net is a encoder-decoder network built with convolutional neural network (CNN). Originally developed for biomedical imaging segmentation task, U-Net has been widely adopted in various image-related work beyond its original scope. With the emergence of video processing models, we also adapt U-Net in our work on driving video generation. For now, we review the basics of U-Net.

U-Net is composed of a cascading down-sampling (contracting) path that encodes the original image into a code vector, and a cascading up-sampling (expansive) path that decodes the code vector back to an image. This contracting and expanding paths makes a "U" shape as shown in Figure 2.4, giving the network its name.

The contracting path performs encoding through the use of a series of convolutional layers with ReLU activations and pooling operations that shrinks the spatial dimensions of the

input image while increasing the depth of feature maps. During this process, the contextual information and important features of the original input image is distilled, and eventually results in a informative code vector.

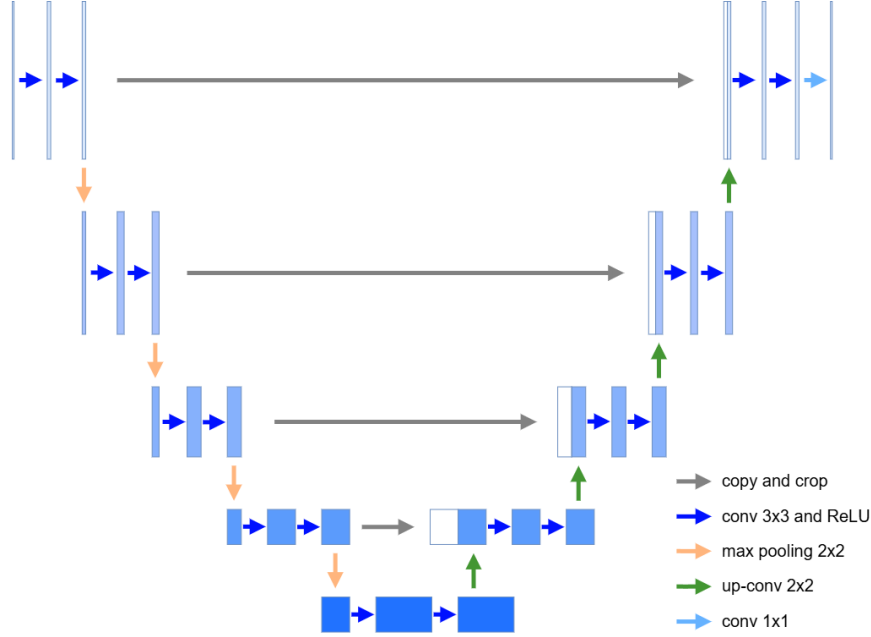


Figure 2.4 U-Net architecture

The expansive path takes the code vector as input (along with the output from the output passed by the contracting path CNNs on the same level), and up-sample the feature maps back to the original image size. This is done by the use of cascading up-convolutions and concatenation operations. The up-convolutions restore the spatial dimensions, while the concatenation operations merge the output from the contracting path with those in the expansive path. This merging process ensures that the network leverages both low-level spatial information and high-level contextual information simultaneously.

U-Net also includes skip connections between the contracting path and the expansive path at each level. This prevents the loss of important information during the abstraction process.

For the original paper which perform segmentation task, the final layer of the U-Net employs a 1x1 convolution to map the multi-channel feature map to the number of classes in the segmentation task.

U-Net is a crucial component for image diffusion models [37] and its 3D variant is used in video diffusion models [38]. For this reason it is included in the review. We provide some more details on the 3D U-Net in the next part and they are discussed again in Chapter 4 where it is used as part of our video generation model.

2.4.1 3D U-Nets

3D U-Net, or temporal U-Net, is a modified U-Net introduced by Blattmann et al. [38] in their video generation work to accommodate the temporal nature of videos.

The key modification is the addition of temporal layers between the convolution layers. Specifically, the convolution layers work just as they do in the regular U-Net, treating the entire video as a big image with dimension $(B \cdot T) \times C \times H \times W$ which stands for Batch, Time, Channel, Height, Width. After each convolution layer, a temporal layer is added that reshapes the output from the spatial convolution layer back into a video format, a tensor with dimension $B \times T \times C' \times H' \times W'$ which treats the Batch and Time as separate dimensions, before processing it. In other words, this temporal layer provides the network with temporal awareness which was previously lacking in the spatial layers. This key modification allows the final 3D U-Net to become networks effective at processing video data.

2.5 Diffusion Models

In recent years, Denoising Diffusion Probabilistic Models (DDPMs) [39, 37, 40], or more commonly referred to as diffusion models, have made significant advances in producing high fidelity image generations. Diffusion models produces high-quality samples by transforming samples from noise distribution into structured data through a series of iterative steps.

Diffusion models is made of two main processes: a forward diffusion process and a reverse denoising process.

2.5.1 Forward Diffusion Process

The forward process is a Markov chain that aims to perturb the data into complete noise one step at a time. It starts with the original data and at each time step, it adds Gaussian noise to the data from the previous step, gradually obscuring the data with noise. Over multiple steps, it transforms the data into a noise-like distribution. The aim is to make the data distribution at the end of the forward process resemble a simple, tractable distribution like Gaussian noise. Specifically, The forward diffusion process defines a sequence of latent variables $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_T$, where $\mathbf{x}_0$ is the original data and $\mathbf{x}_T$ approximates Gaussian noise. This process can be described as:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad (2.19)$$

where at each step, q adds Gaussian noise according to a variance schedule hyperparameter $\beta_{1:T}$.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{(1 - \beta_t)}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad (2.20)$$

As noted by Sohl-Dickstein et al. [39], this selection of q allows us to marginalize the joint distribution in Eq. 2.19 and arrive at an analytical form expression for the noisy data at time step t given only the original data (at time step 0). Specifically, with $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=0}^t \alpha_s$, we have:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}) \quad (2.21)$$

This expression enables sampling of $\mathbf{x}_t$ by drawing $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon} \quad (2.22)$$

2.5.2 Reverse Denoising Process

In contrast, the reverse process aims to reconstruct the original data from Gaussian noise. It begins with a sample from the noise distribution and iteratively denoises it to recover the original data. In training, this process is guided by a neural network trained to predict the noise difference at each step, effectively learning the reverse of the forward diffusion process. In doing so, the model learns how to remove the added noise at each time step, and by iteratively removing noise it is able to retrieve the original data. At test time, the reverse process leverages the learned conditional distributions to generate new samples that closely resemble the original data, starting from random noise.

Mathematically, the reverse process recovers the original data by learning the reverse conditional distributions:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \quad (2.23)$$

Here, θ is the neural network parameters; μ_θ and Σ_θ are the mean and variance predicted by the neural network at each step. It is crucial for the denoising network to learn to match the time-reversed forward process in order for it to generate high-quality data. That is, the reverse joint distribution $p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ needs to approximate the forward joint distribution $q(\mathbf{x}_{0:T}) = q(\mathbf{x}_0) \prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1})$. To ensure this, the model is

trained to minimize the Kullback–Leibler (KL) Divergence between the two joint distributions, $KL(p_\theta(\mathbf{x}_{0:T})||q(\mathbf{x}_{0:T}))$, and a training objective of variational lower bound (VLB) on the data likelihood is used to achieve this goal:

$$L_{\text{VLB}} = \mathbb{E}_q \left[\sum_{t=1}^T D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1) \right] \quad (2.24)$$

2.5.3 Stable Diffusion

An important development in diffusion based models is the Stable Diffusion introduced by Blattmann et al. [41]. It contains several important modifications which are the basis for the Stable Video Diffusion [42], a video model that we later build upon in our Ctrl-V work in Chapter 4.

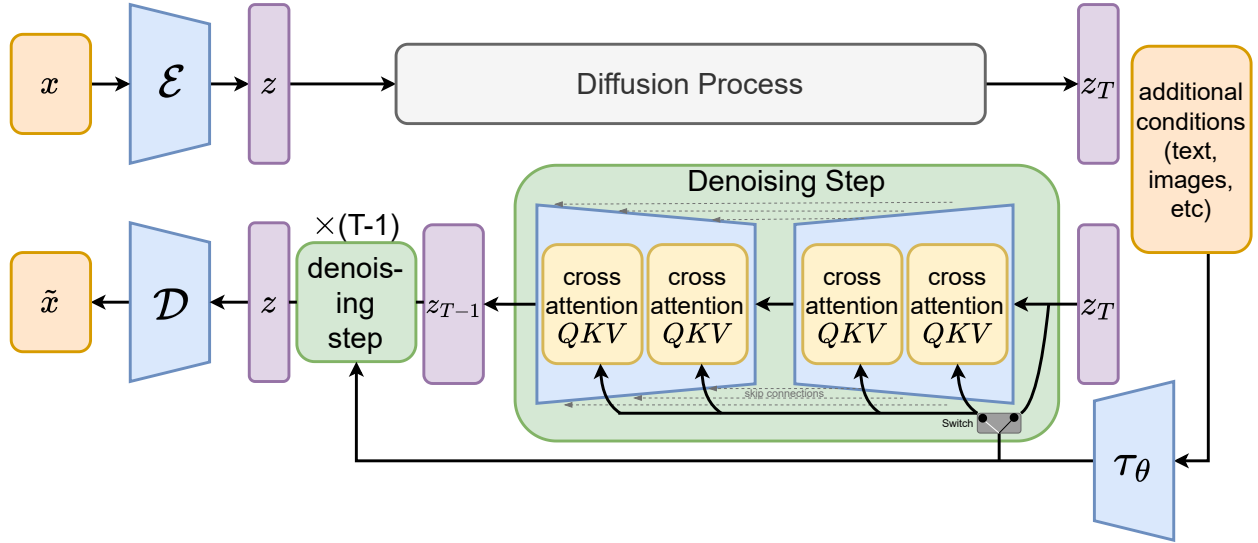


Figure 2.5 Stable Diffusion model architecture

As seen in Figure 2.5, the model operates in a latent space instead of the image space. Stable Diffusion does this by introducing an image encoder $\mathcal{E}$ and an image decoder $\mathcal{D}$. Input images, $\mathbf{x}$, passes through the image encoder to be transformed into latent representation $\mathbf{z}$, before going through the diffusion process. After the forward process (which adds T steps of noise to transform $\mathbf{z}$ to $\mathbf{z}_T$) and reverse process (which transforms $\mathbf{z}_T$ back to $\mathbf{z}$), the image decoder reconstruct the latent representation back into an image.

The reverse process is parameterized by a U-Net. Crucially, the U-Net is expanded with cross attention capacity in order to incorporate condition (or control inputs). Assuming the control input $\mathbf{y}$ is processed by an adapter network τ_θ . In order to mend this control signal

$\tau_\theta(\mathbf{y})$ into the denoising process, a set of cross attention parameters ($\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$) is added in each layer of the U-Net. They map the control signal into the intermediate layers of U-Net through $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right) \mathbf{V}$ where at each layer i :

$$\mathbf{Q} = \mathbf{W}_Q^{(i)} \cdot \varphi_i(z_t) \quad \mathbf{K} = \mathbf{W}_K^{(i)} \cdot \tau_\theta(\mathbf{y}) \quad \mathbf{V} = \mathbf{W}_V^{(i)} \cdot \tau_\theta(\mathbf{y}) \quad (2.25)$$

Here, $\varphi_i(z_t)$ is the intermediate representation output by the i -th U-Net layer.

These modifications makes Stable Diffusion a state of the art image generation model. It further opens the gate to better controllable generation of images, and leads to development such as ControlNet [43], which will be reviewed in our Ctrl-V work.

CHAPTER 3 VIDEO PREDICTION VIA SPATIO TEMPORAL EXTERNAL ATTENTION AND MEMORY

Conditional generation is a basic form of controllable generation. In particular, generating video frames using prior frames as conditioning is a well known method in video prediction tasks. Recently, video generation has been dominated by diffusion methods, and traditional non-diffusion based models are often overlooked.

Specifically, predicting long sequences of high-quality video with simple, low-cost models remains a challenging task. While recent advancements like SimVP have achieved significant improvements in generation quality using block-to-block methods with convolutional neural networks (CNNs), eliminating the need for frequent frame-level processing, block-based prediction still struggles to generate coherent lengthy videos. To address this, we propose a new model, ViP-STEAM, and introduce three key innovations. We shift the generator’s conditioning from previously generated frames to pre-generation representations, reducing encoding inaccuracies and compounding errors. Our STEAM blocks, a novel CNN-based network block with learnable memory, enhance long-term information preservation, counter-acting information loss over time. Additionally, our memory update mechanism enables the model to control what information to retain or discard, allowing for improved video prediction capabilities. We demonstrate the effectiveness of our approach on benchmark datasets moving-MNIST, KTH, CityScape, and BAIR

In this collaborative work, my contribution are as follows:

- Contribution to the inception and formulation of the work.
- Consistent refining of model architecture.
- Multiple designs of the memory mechanism design and implementation
- Partial design of the STEAM blocks.
- Running experiments on the datasets.
- Implementation and running of evaluation.
- Visualization of results.

This work is in the process of being submitted to the IEEE/CVF Winter Conference on Applications of Computer Vision 2025, and we plan to submit it onto arxiv as well. Much of the work presented here will overlap with the arxiv version.

3.1 Introduction

The ability to predict future events based on visual inputs, akin to watching a video, is a vital skill for humans and animals alike. This capability allows us to anticipate and prepare for upcoming events, enhancing our survival and decision-making abilities. As technology advances, video prediction is becoming increasingly crucial in various applications, such as robotics and autonomous driving, which rely on accurate forecasting of visual data. In the realm of video prediction, current research are dominated by diffusion models and in general, large size models. Light weight and simplistic models are often out-matched. In this work, we aim to push the limit of simplistic models to achieve better video prediction results.

In terms of light-weight models, work by Chiu et al. [44], Shouno [45], Xu et al. [46] explore the use of convolutional neural networks (CNNs) without recurrent neural networks (RNNs) or long short-term memory (LSTM) networks. Although these CNN-only models have shown promise, their simplicity often leads to limitations, requiring compromises in architecture or training. For instance, some models rely on pixel flowing, where pixel values are borrowed from nearby voxels [47], or employ a two-part framework to separate motion-specific propagation and motion-agnostic generation [48].

Recently, SimVP [4, 3] has established itself as a state-of-the-art model, showcasing competitive performance with a simple CNN-based encoder-decoder architecture, enhanced by a CNN translator model. This design achieves impressive results on video prediction benchmarks, highlighting the potential of CNN-only models in this domain.

However, despite this progress, simplistic non-diffusion models require further architectural advancements to rival the performance of cutting-edge diffusion models. While SimVPv2, an improved version with gated spatiotemporal attention (gSTA) units [3], has made significant strides, it still faces three major limitations:

1. **Limited context:** In SimVP’s block-to-block approach, the output block is only conditioned on the input block without any history or stored information of frames prior to the input frame block. This effectively limit the generation context to the length of the frame block, which can cause the model to discard important information and generate frames inconsistent with earlier frames.
2. **Images degradation:** SimVP’s generation process beyond its context length involves sequential decoding and re-encoding of generated images, leading to compounding errors. Inaccuracies from each output frame are accrued in the next block. The cumulative error becomes more and more noticeable in each of the decoding and re-encoding

loop.

3. **Lack of stochasticity:** SimVP lacks the ability to capture the inherent randomness and uncertainty in video data. This leads to blurry predictions in situations with multiple possible outcomes. For instance, at an intersection, a car may proceed straight, turn left, or turn right, but SimVP’s reliance on expected future frames results in indistinct predictions, failing to account for the various possibilities.

To tackle these existing problems in the state-of-the-art model, we introduce **V**ideo **P**rediction via **S**patio **T**emporal **E**xternal **A**ttention + **M**emory (ViP-STEAM), which improves on SimVP while maintaining a small training and memory footprint. All of our experiments are done using single RTX-8000 GPU, and a full model takes less than a week to train on the single GPU. Furthermore, our model retains the a simple CNN-based structure. Overall, our contributions are as follows:

1. We incorporate *Gated Spatio-Temporal External Attention Memory (STEAM)* blocks to enhance the model’s long-term memory capabilities. These blocks provide a way to store past features outside of the generation block. This allows our model to to access and utilize previous features, even after generating new frames by selectively attending to relevant past information. This way, our model can better capture complex spatial and temporal relationships.
2. We implement what we call *feature recycle generation*, which essentially moves the block generation process (of frames) from pixel level to feature level. This bypass the decoding and re-encoding of frames during block generation, and in turn reduces compounding errors in the generated images.
3. We compare with not only the common and state-of-the-art CNN models, but also recent state-of-the-art diffusion based model MCVD and show that ViP-STEAM trained on a single GPU within 1 week can produce competitive results on popular metrics for widely-used benchmarks.

3.2 Related Work

Traditionally, video prediction is often tackled by using recurrent neural networks (RNNs and LSTMs) in combination with CNNs [16, 15, 17]. For example, ConvLSTM [16] adds convolution structure to LSTMs to enable image-based input and state-to-state transition. PredRNN [15] builds on it and introduces the idea of spatiotemporal memory flow and a novel

recurrent unit called spatial-temporal LSTM (ST-LSTM); this is to decouple the long-term and short-term memory of the recurrent structures and to mitigate the problem of important information being forgotten at the tail end of long sequence generation. Stochastic Video Generation (SVG-LP) [49] employs a basic convolutional LSTM structure, and learns a prior that captures stochasticity at every time step of the generation process. Hierarchical vRNN [50] introduces the idea of multi-level latent where, at each step, a densely connected hierarchy of latent variables are injected into the generation process. E3D-LSTM [17] explores the use of 3D CNN as a basis for an encoder-decoder structure while maintaining the video state using a recurrent structure; this is said to encourage the storage of better short term features while making the RNNs motion-aware. Models such as PhyDNet [51], CrevNet [52], and more all employ this approach of using RNN/LSTM along with CNN, albeit with increasingly sophisticated architecture like an invertible 2-way autoencoder [52] or training method like teacher-student distilling [44].

Memory-based method has also been explored in work by Lee et al. [53]. However the memory is purely based on initial frames and never updated according to the generated frames. In long term sequence generation where the scene is changing (driving video etc.) this type of memory becomes less useful.

Aside from these methods, recent developments in Denoising Diffusion Probabilistic Models (DDPMs) [54, 39] has enabled a new approach to video prediction. Diffusion models have surpassed existing GAN-based methods in image synthesis [55]; this prompted diffusion-based video prediction models such as Masked Conditional Video Diffusion (MCVD) [56] that generate blocks of video frames through the diffusion process. Training diffusion models usually incur large time and space overhead, and even though MCVD aims to minimize these costs, it still takes 4 GPUs 1 week to train the model while cutting down the number of input frames on a lot of the benchmark tests.

3.3 Method

3.3.1 Model Architecture

In line with SimVP [4, 3], our model consists of three primary components: a spatial encoder, a spatial decoder, and a translator. The spatial encoder preprocesses input data to be used by the translator, which takes care of temporal translation in feature space. The output is then passed to the spatial decoder to be reconstructed into frames.

We have developed two variants of our model: the CVAE-Trans variant and the STEAM-Trans variant. The CVAE-Trans variant’s architecture is illustrated in Figure 3.1. The

process begins with the spatial encoder converting input and target frames into corresponding feature blocks. Next, the latent encoder captures the latent distribution from these features, followed by the reparameterization trick being employed to sample a latent feature from this distribution. The latent decoder then processes the sampled feature and input frame features to produce predicted features, which are subsequently reconstructed into frames by the spatial decoder. The various losses employed during our training process are annotated on the right side of the figure. Detailed descriptions of the variants and spatial encoder and decoder modules are provided in the following sections.

Spatial Encoder (F) and Decoder (F^{-1})

SimVP [4, 3]’s foundational structure contain a pair of spatial encoder and decoder, both of which consists of N_s convolutional blocks. In our work, we retain this design. Specifically, each block is made of a 2d convolution layer, a group normalization layer, and a nonlinear activation function. The encoder serves to reduction dimensionality, where every other convolutional layer has a doubled stride size, effectively downsampling the feature maps. In contrast, the decoder utilizes a pixel-shuffle operation in every second convolutional layer to upsample the data and restore resolution. Additionally, we incorporate an input-reconstruction loss function to ensure that the spatial encoder and decoder operate within the same spatial domain, promoting consistent representation and accurate reconstruction of the input data.

Translator

We create two variants of our model: one employing a CVAE-translator, we call this variant CVAE-Trans, and the other featuring a STEAM-translator we call this model STEAM-Trans .

CVAE-Trans: This variant utilizes a Conditional Variational Autoencoder (CVAE) [57]. The structure of the latent encoder and decoder are shown in Figure 3.2. The latent encoder outputs parameters for the isotropic Gaussian latent posterior distribution from both the target and context frames while the latent decoder samples from this distribution, and use these samples along with context features to predict target frame features. The latent encoder (Q) and decoder (P) are composed of a sequence of Gated Spatio-Temporal External Attention Memory (STEAM) blocks, detailed in the subsequent section.

STEAM-Trans: In this variant, the translator do not use any CVAE structure. It is simply made up of a sequence of STEAM blocks.

STEAM Blocks

Taking inspiration from gSTA blocks used in the SimVP+gSTA framework [3], we create an improved Spatio-Temporal External Attention Memory blocks, or STEAMblocks. A compar-

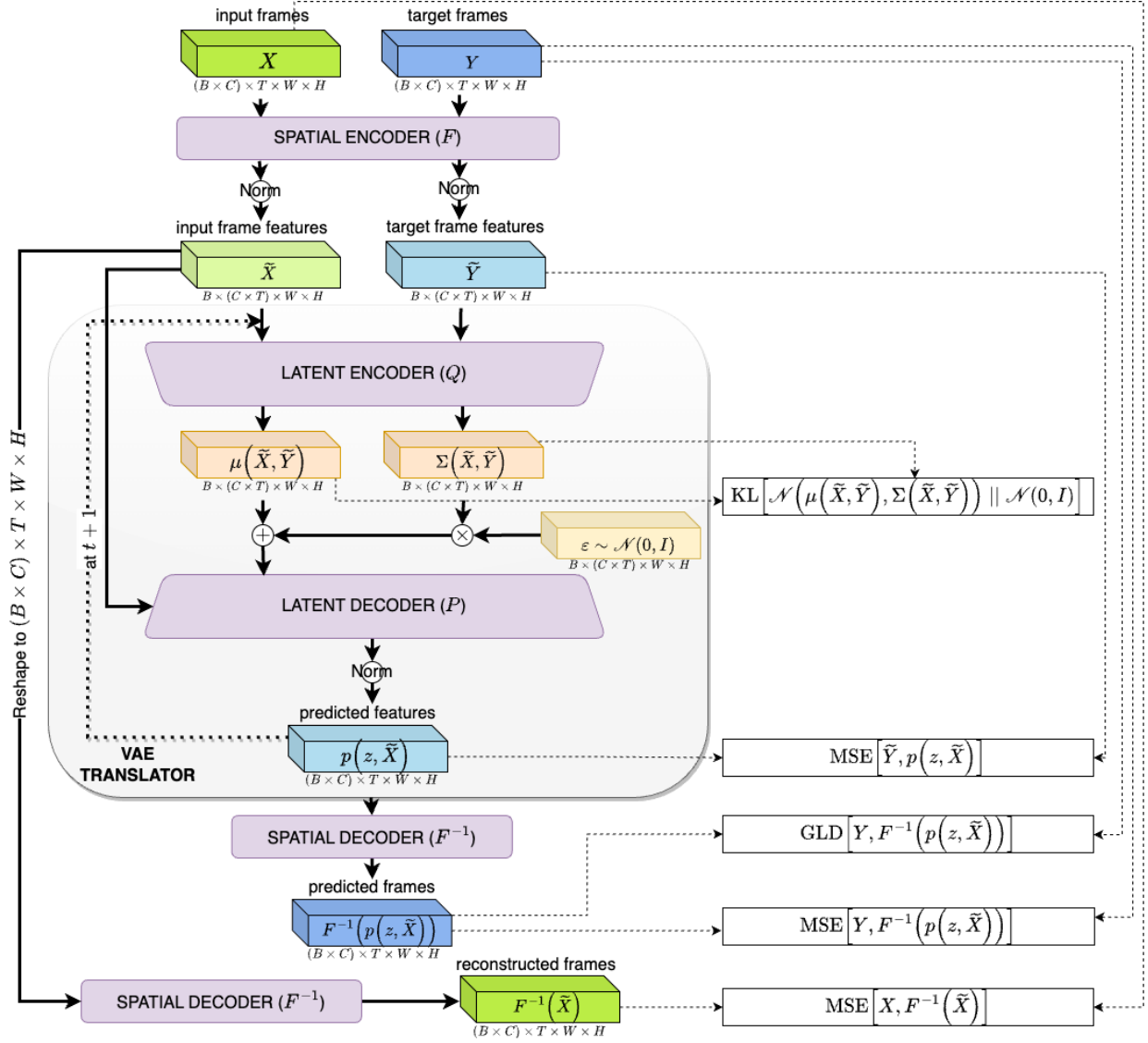


Figure 3.1 The ViP-STEAM CVAE-Trans Framework: the generation process within the ViP-STEAM CVAE-Trans network.

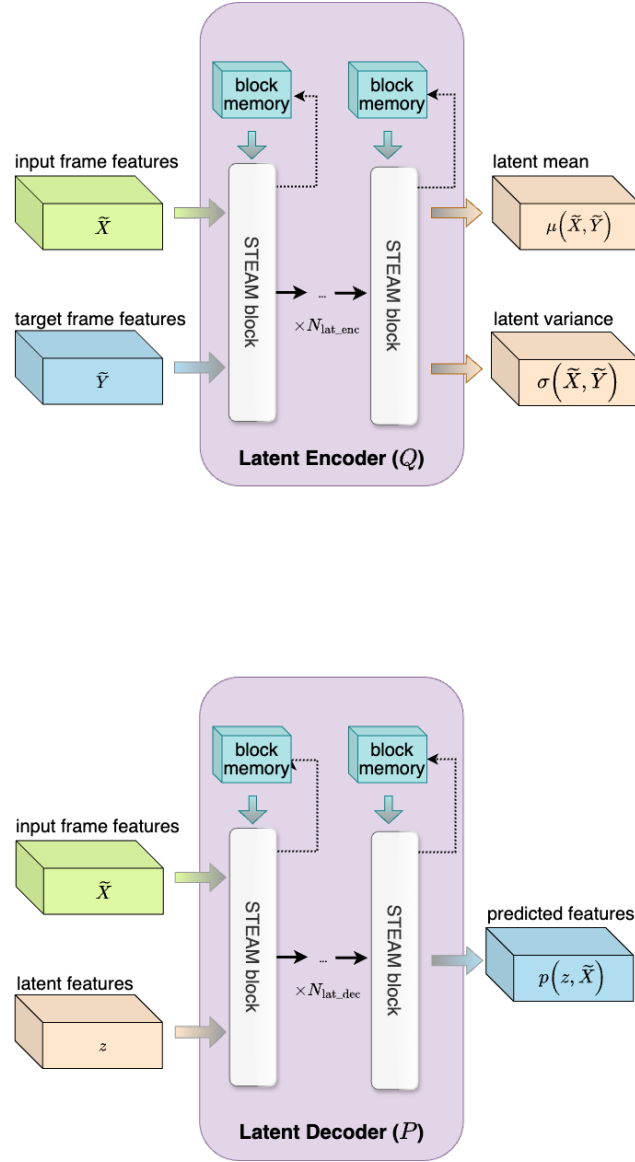
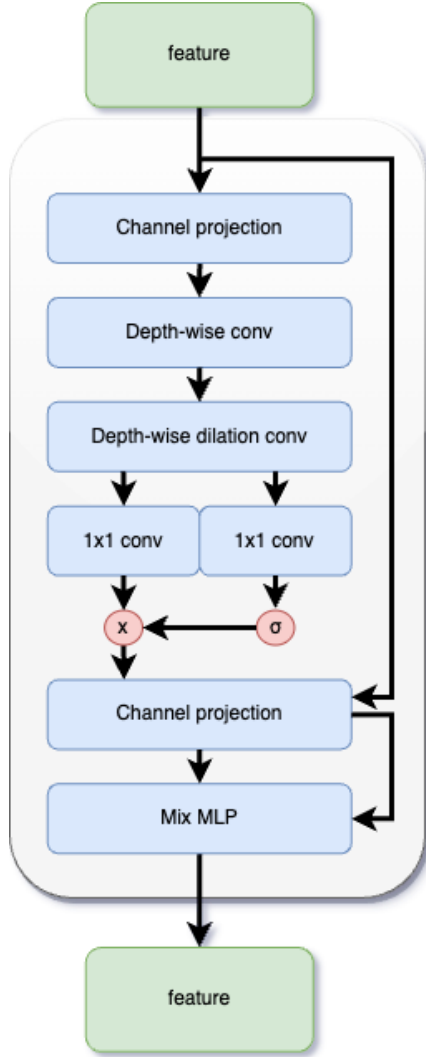
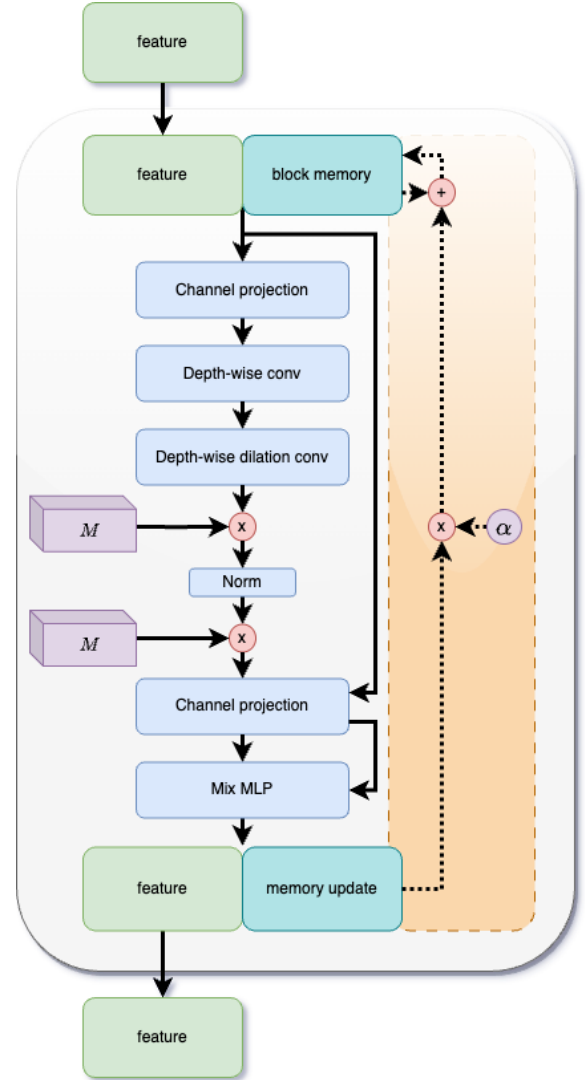


Figure 3.2 Architecture of the latent encoder and decoder.



(a) gSTA-block.



(b) STEAM block.

Figure 3.3 Comparison of the gSTA-block [3] and the STEAM block

ison between the internal structure of gSTA blocks and STEAM block are shown in Figure 3.3. The most important novelty is its inclusion of an memory block and a new memory update mechanism. In the next sections, we dive into the memory aspect of STEAM blocks first, before elaborating more on its structure.

STEAM Blocks: Memory Update

The STEAM blocks operate by receiving two inputs: the output from the preceding block and the stored memory from the previous time step. The output from the STEAM blocks is then divided into two components: one segment serves as the input for the subsequent STEAM block, while the other is retained for updating the memory state of the current block.

The memory state update process involves adding the output memory update tensor to the block’s existing memory, providing additional residual information. To ensure stability during learning, we employ the ReZero technique [58] on the memory update’s skip connection. Mathematically, the update step can be represented as:

$$\text{block_mem}_{t+1} = \text{block_mem}_t + \alpha \times \text{mem_update}_t \quad (3.1)$$

where α is initialized to 0 at the start of training. Figure 3.3 illustrates this update process in the orange highlighted box.

Additionally, the block memory of each STEAM block is a learnable vector concatenated to the feature vector which is only updated by the memory update block when there are recurrent predictions. The M matrices are external memories. They are implemented using learnable global tokens, following [59].

STEAM Blocks: Inner Structure

The structure of the blocks are shown in Figure 3.3. In this section, we trace the steps of an feature tensor and explain the block components along the way. As feature tensors enter a STEAM block, it is first met by three convolutional layers, each serving an unique purpose:

1. **Depth-wise Convolution with a Single Channel kernel:** This layer captures local receptive fields, and process spatial information.
2. **Depth-wise Dilation Convolution with a Single Channel Kernel:** This layer extends the scope to include surrounding local receptive fields, effectively broadening the network’s spatial understanding.
3. **1×1 Convolutional Layer:** This layer aggregates information across all channels.

These layers employ the separation convolution technique [60, 61], a widely adopted strategy [3, 62, 63], to approximate the effect of large kernel convolutions while reducing parameter usage. This technique involves restructuring large kernel convolutions to decrease computational demand. Recent studies [64, 65] have demonstrated that CNNs with larger kernels exhibit improved shape bias over texture in their extracted features, enabling the network to capture more comprehensive spatial features due to the broader receptive fields of these kernels.

After the separable convolution layers, the transformed feature tensors are now processed by external attention layers. These layers compute attention between the input features $\hat{\mathbf{X}}_{\text{in}} \in \mathbb{R}^{(T \times C) \times (H \times W)}$, and an external memory unit $\mathbf{M} \in \mathbb{R}^{S \times (T \times C)}$. The mathematical formulation of the attention computation can be presented as follows:

$$\mathbf{A} = (\alpha)_{i,j} = \text{Norm}(\hat{\mathbf{X}}_{\text{in}}^{\top} \mathbf{M}^{\top}), \quad (3.2)$$

$$\hat{\mathbf{X}}_{\text{out}} = (\mathbf{A} \mathbf{M})^{\top}. \quad (3.3)$$

Equation 3.2 calculates the attention map $\mathbf{A}$, where $\alpha_{i,j}$ represents the affinities between the input vector and the external memory. Equation 3.3 computes the output features, $\hat{\mathbf{X}}_{\text{out}}$, by taking the product of the attention map $\mathbf{A}$ and the external memory $\mathbf{M}$.

This external attention mechanism has been shown to effectively capture global context [59] by treating the external memory as a repository of global information. By attending to this memory, the network can preserve and model important information over extended sequences. Notably, the external attention mechanism is also recognized for its efficiency [59]. In the gSTA block, the gated spatio-temporal attention layers comprise two 1×1 convolutional layers, a sigmoid function, and element-wise multiplication, totaling $2C^2$ parameters. In contrast, STEAM block’s external attention layers contain just $C \times S$ parameters. As observed by Guo et al. [59], a small value of $S = 64$ can achieve good empirical results while significantly reducing the number of parameters ($S < 2C$).

3.3.2 Feature Recycling

Our model conforms to the purely CNN-based architecture described by Tan et al. [3], generating a sequence of N frames in a single forward pass. To extend the generation beyond N frames, traditional models typically employ an iterative process, reintroducing the predicted output as input for subsequent predictions, as depicted in Figure 3.4. In our work, we propose to transition the iterative generation process to the feature space. This adjustment reduces computational complexity and the propensity for error accumulation due to repeated

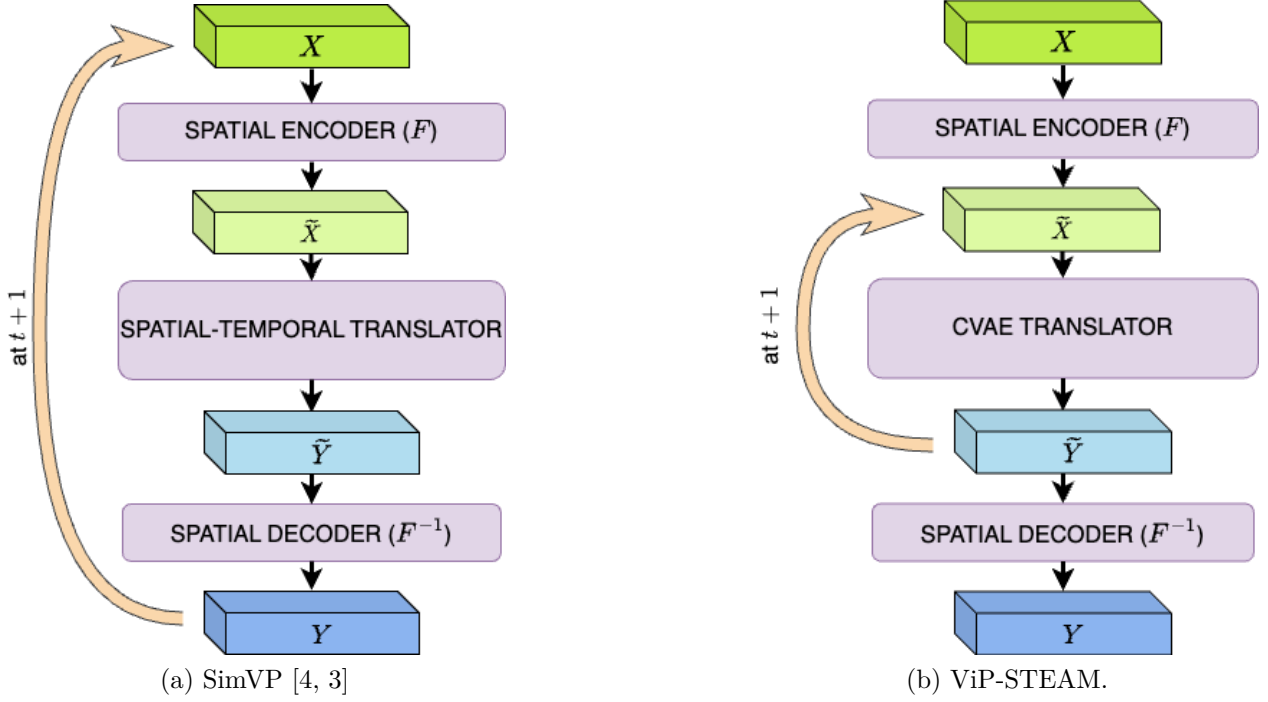


Figure 3.4 Iterative generation loops. The left Figure illustrates the original iterative process as described in SimVP [4, 3]. The right Figure depicts the modifications we have implemented in our approach.

encoding. Our modified iterative generation process is illustrated in Figure 3.4. To facilitate effective feature recycling for future predictions, we implement the following modifications:

Skip Connection Removal: We omit the skip connection between the initial layer of the spatial encoder and the final layer of the spatial decoder. This architectural modification eliminates the need for repeated encoding during each iteration of the generative process, streamlining the model’s operations and enhancing its efficiency.

Input-Reconstruction Loss Addition: We train the model with an additional input-reconstruction auxiliary loss. This is often used to train auto-encoders, and we include this loss to encourage the decoder to accurately reconstruct the original input from the encoded features, thus satisfying the condition $F^{-1}(F(\mathbf{X})) = \mathbf{X}$.

Introduction of Feature-Prediction Loss: A feature-prediction loss has been integrated into our learning objective. This ensures the translator’s input and output features are aligned in the same space, adhering to the condition $\tilde{\mathbf{Y}} = P(\mathbf{z}, \tilde{\mathbf{X}})$ where $\tilde{\mathbf{Y}} = F(\mathbf{Y})$, $\tilde{\mathbf{X}} = F(\mathbf{X})$ and $\mathbf{z} \sim \mathcal{N}(\mu(\tilde{\mathbf{X}}, \tilde{\mathbf{Y}}), \Sigma(\tilde{\mathbf{X}}, \tilde{\mathbf{Y}}))$.

Spatial Features Normalizing: We incorporated normalization layers subsequent to the spatial encoder and latent decoder. This ensures that the spatial features of all frames maintain a consistent magnitude, and prevents the embedding features from degenerating into zero vectors due to the feature-prediction loss.

3.3.3 Training Objectives

In this work, our model is trained using 5 different losses, we list them and explain their purposes:

1. **Target Frame Prediction Loss:** This loss function measures how well the model predicts target frames.

$$\mathcal{L}_{\text{frame_pred}}(\mathbf{X}, \mathbf{Y}) = \text{MSE}\left[\mathbf{Y}, F^{-1}\left(p(z, F(\mathbf{X}))\right)\right] \quad (3.4)$$

2. **Input Frame Reconstruction Loss:** This loss function measures how accurately the model can reconstruct input frames.

$$\mathcal{L}_{\text{input_recon}}(\mathbf{X}, \mathbf{Y}) = \text{MSE}\left[\mathbf{X}, F^{-1}\left(F(\mathbf{X})\right)\right] \quad (3.5)$$

3. **Target Feature Prediction Loss:** This loss function measures the accuracy of the model in predicting target features.

$$\mathcal{L}_{\text{feature_pred}}(\mathbf{X}) = \text{MSE}\left[F(\mathbf{Y}), p\left(z, F(\mathbf{X})\right)\right] \quad (3.6)$$

4. **Gradient Difference Loss:** This loss function enhances the sharpness of prediction by imposing penalties on the discrepancies of the gradients of the predicted images and those of the target images. The concept was first introduced in video prediction tasks in [66].

$$\mathcal{L}_{\text{gdl}}(\mathbf{X}, \mathbf{Y}) = \text{MSE}\left[\frac{\partial \mathbf{Y}}{\partial i}, \frac{\partial \hat{\mathbf{Y}}}{\partial i}\right] + \text{MSE}\left[\frac{\partial \mathbf{Y}}{\partial j}, \frac{\partial \hat{\mathbf{Y}}}{\partial j}\right] \quad (3.7)$$

where $\hat{\mathbf{Y}} = F^{-1}\left(p(z, F(\mathbf{X}))\right)$ and $\frac{\partial \mathbf{Y}}{\partial j} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} * \mathbf{Y}$ is the vertical image gradient. Similarly, $\frac{\partial \mathbf{Y}}{\partial i}$ is the horizontal image gradient.

5. **CVAE Translator’s Kullback-Leibler(KL) Divergence Loss:** This loss function helps the latent encoder align its latent distribution prediction more closely with a

standard multivariate Gaussian distribution.

$$\mathcal{L}_{\text{KL}}(\mathbf{X}, \mathbf{Y}) = \text{KL} \left[\mathcal{N}(\mu(\tilde{\mathbf{X}}, \tilde{\mathbf{Y}}), \Sigma(\tilde{\mathbf{X}}, \tilde{\mathbf{Y}})) \parallel \mathcal{N}(0, I) \right] \quad (3.8)$$

3.4 Experiments

In this section, we provide an overview of the experiments conducted in our study. In Section 3.4.1, we present the datasets used in this work. In Section 3.4.2, we outline the methodology of the pretraining process. In Section 3.4.3 we provide details on our implementation. In Section 3.4.4 we discuss our findings.

Table 3.1 Summary of Dataset Statistics.

Name	Resolution	T	T'
Moving-MNIST [67, 68]	1, 64, 64	10	50
KTH-40 [69]	1, 128, 128	10	40
CityScape [70]	3, 128, 128	5	25
BAIR [71]	3, 64, 64	4	12

3.4.1 Datasets

In this study, we concentrate on video prediction tasks involving extended sequences of target frames, specifically scenarios where the number of prediction frames (T') exceeds the number of context frames (T). This criterion enables us to assess our network’s capabilities in iterative prediction.

We evaluate our model on four benchmark datasets: **Moving-MNIST** [67, 68] and **KTH-40** [69], which are commonly used by state-of-the-art RNN and CNN-based video prediction networks, including SimVP. Additionally, we experiment on **CityScapes** [70] and **BAIR** [71], which feature more complex videos with diverse scenery, actions, and object interactions. Although these datasets are not typically reported on by similar RNN and CNN-based networks, we aim to explore the limits of our lightweight, purely CNN model.

Each dataset has unique resolution and task definitions, with varying numbers of context frames (T) and prediction frames (T'). The specific configurations for each dataset are outlined in Table 3.1. We employ a comprehensive evaluation metric suite, comprising MSE, PSNR, SSIM [73], and LPIPS [74], to assess the performance of our model.

Table 3.2 Performance comparison with popular CNN-based models. The two variants of ViP-STEAM models used in the experiments are labeled in their names. We present results with the better of the two variants for each tasks – both variants are shown in the case of KTH-40 since they perform similarly. Their implementation details can be found in Section 3.3.1. The ViP-STEAM models marked with † are trained using the dataset configuration outlined in Table 3.1 and evaluated with the configuration specified in this table. We used padding to adjust any input dimension misalignments. ‡ The KTH-40 dataset employed in MCVD has a frame resolution of 64×64 , which is different from the resolution used in other studies. Prior work has evaluated methods using different combinations of past conditioning frames and future frames for prediction. We select past and future frame combinations which emphasize making long predictions in the future in such a way that we can compare with many prior methods providing results for these configurations.

Task	T	T'	Method	MSE↓	PSNR↑	SSIM↑	LPIPS↓
MMNIST [68, 67]	10	50	ConvLSTM [16]	67.60	-	0.816	-
	10	50	PredRNN [15]	70.11	-	0.882	-
	10	50	E3d-LSTM [17]	157.18	-	0.781	-
	10	50	SimVP+gSTA [3]	77.80	-	0.795	-
	10	50	Ours (STEAM-Trans)	56.80	-	0.837	-
KTH-40 [69]	10	40	ConvLSTM [16]	-	22.85	0.639	-
	10	40	PredRNN [15]	-	24.16	0.703	-
	10	40	E3d-LSTM [17]	-	27.24	0.810	-
	10	40	SimVP+gSTA [3]	-	33.35	0.895	0.350
	10	40	MCVD ‡ [56]	-	26.40	0.812	-
	10	40	Ours (STEAM-Trans)	-	32.12	0.895	0.345
	10	40	Ours (CVAE-Trans)	-	31.59	0.897	0.336
CityScapes [70]	2	28	SVG-LP [49]	-	-	0.574	0.549
	2	28	vRNN 1L [50]	-	-	0.609	0.304
	2	28	Hier-vRNN [50]	-	-	0.628	0.264
	2	28	GHVAE [72]	-	-	0.740	0.193
	2	28	MCVD [56]	-	-	0.690	0.112
	2	28	Ours (CVAE-Trans) †	-	31.11	0.773	0.353
	5	25	Ours (CVAE-Trans)	-	31.34	0.790	0.340
BAIR [71]	2	28	ConvLSTM [16]	-	20.52	0.807	-
	2	25	Hier-vRNN [50]	-	-	0.822	0.055
	2	25	SVG-LP [49]	-	-	0.816	0.061
	2	28	PredRNN.v2 [15]	-	22.74	0.861	-
	2	28	MCVD [56]	-	17.70	0.797	-
	2	28	Ours (CVAE-Trans) †	193.32	31.12	0.835	0.095
	4	12	Ours (CVAE-Trans)	157.29	31.87	0.868	0.091

3.4.2 Pretraining

We pretrain our models on shorter generation tasks before training them on the main long sequence generation tasks. We refer to this stage as the pretraining phase. During this phase, the model is configured to predict an equal number of output frames as the number of input frames, without any iterative generation steps. For example, in the KTH-40 task, which typically involves generating 40 frames from 10 given conditional frames, we adapt the task to generate 10 frames from 10 input frames during the pretraining stage. This approach enables the model to develop essential skills and knowledge, which are later refined for the more complex long sequence generation tasks.

3.4.3 Implementation Details

Our experimental framework is built upon OpenSTL, an open-source codebase developed by Tan et al. [75] for video prediction benchmarks. We leverage the SimVP implementation and evaluation code provided in this resource to ensure precise and standardized testing of our model. Furthermore, we utilize the repository’s code to conduct baseline evaluations of ConvLSTM, predRNN, and E3d-LSTM on the Moving-MNIST 10 $\rightarrow$ 50 configuration, enabling a comprehensive comparison of our approach with established methods.

3.4.4 Result Analysis

We conduct experiments on all datasets listed in Table 3.1, employing the methods outlined in preceding sections. Additionally, we implement baseline models comprising prominent CNN/RNN-based architectures [76, 4, 3, 56] for comparative analysis. Furthermore, we incorporate scores reported by MCVD [56], a state-of-the-art diffusion model, as a reference

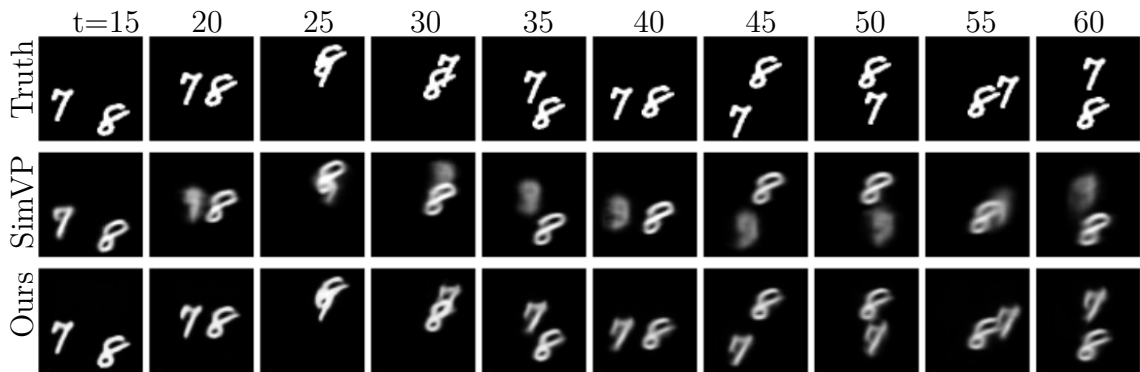


Figure 3.5 Comparing generated results of SimVP versus our model on Moving-MNIST

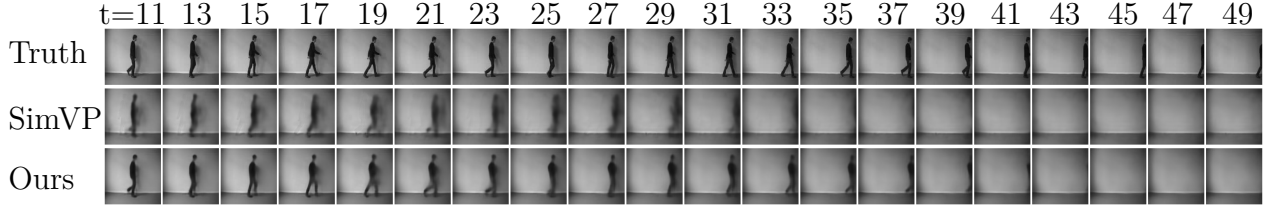


Figure 3.6 Comparison between generated results of SimVP and our model on KTH-40

point. Our results, compiled in Table 3.2, demonstrate notable performance.

On Moving-MNIST, our model exhibits a significant reduction in MSE loss, yielding predictions with enhanced visual quality (Figure 3.5). The generated images display reduced blur, maintaining recognizable digits even at the 60th time step.

On KTH-40, our model surpasses SimVP in SSIM metrics but marginally underperforms in PSNR. However, visual inspection of generated images (Figure 3.6) reveals higher quality predictions, suggesting PSNR’s limitations in capturing distortions like blur [77].

On CityScapes, our model achieves state-of-the-art SSIM scores but falls short in LPIPS and PSNR metrics. As illustrated in Figure 3.7, our model struggles to reconstruct unseen elements, a limitation inherent to non-generative models like ours.

On BAIR, our model attains a higher PSNR score than state-of-the-art models but underperforms in SSIM and LPIPS. The complex robotic arm movements and object interactions in BAIR pose challenges for our model, leading to predictions that minimize MSE loss by omitting the arm (Figure 3.7).

Notably, our simple, purely CNN-based model demonstrates competitive performance on CityScapes and BAIR, highlighting its potential in video prediction tasks.

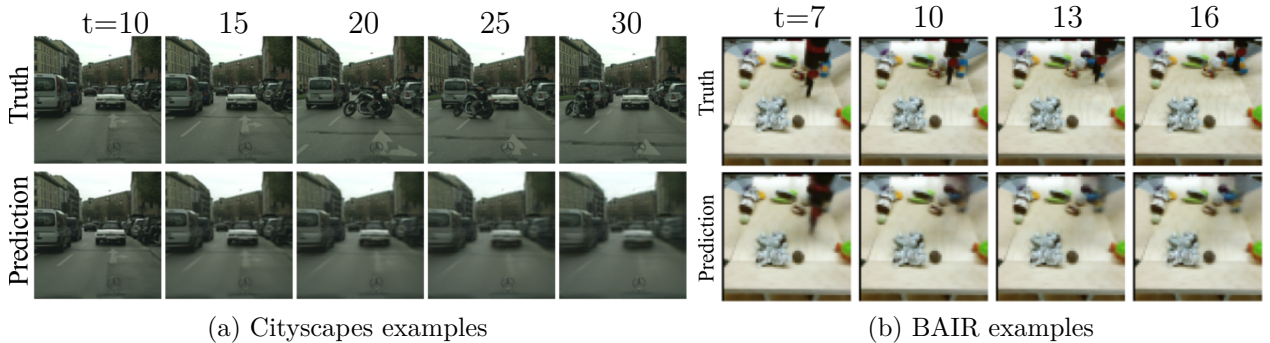


Figure 3.7 Examples of predicted results by ViP-STEAM.

3.5 Ablation Studies

3.5.1 Feature vs. Frame Recycling in ViP-STEAM

Table 3.3 Ablation results for: feature-vs-frame recycling (FR) and STEAM blocks with-vs-without memory update (MU). The second column shows (FR) the status of the feature recycling, and the third column (MU) shows the status of the memory update.

Task	FR	MU	PSNR $\uparrow$	SSIM $\uparrow$
KTH-40	$\times$	$\checkmark$	30.88	0.894
	$\checkmark$	$\times$	31.04	0.896
	$\checkmark$	$\checkmark$	31.59	0.897
CityScape	$\times$	$\checkmark$	31.17	0.785
	$\checkmark$	$\times$	31.32	0.789
	$\checkmark$	$\checkmark$	31.34	0.790
BAIR	$\times$	$\checkmark$	31.62	0.859
	$\checkmark$	$\times$	31.08	0.837
	$\checkmark$	$\checkmark$	31.87	0.868

We analyze the effects of feature recycling and frame recycling on the performance of ViP-STEAM during inference. Through ablation experiments, we compare the network’s performance under two distinct output recycling strategies employed during the iterative generation process: (1) recycling output frames by feeding them back into the spatial encoder, and (2) recycling output features by returning them to the latent decoder. The visual disparities between frame-recycled and feature-recycled generation are illustrated in Figure 3.8, with the frame produced by frame-recycling exhibiting increased blur and lacking background details compared to the more defined frame generated through feature-recycling. This discrepancy is further substantiated by the performance metrics presented in Table 3.3, potentially attributable to information loss incurred during repeated encoding processes.

Our findings suggest that the strategic recycling of generative features for iterative gener-

Table 3.4 Analyzing the utility of STEAM blocks. We compare the performance of SimVP+gSTA model variants versus our ViP-STEAM-Trans approach on MMNIST10 $\rightarrow$ 50.

	# param.	MSE $\downarrow$	MAE $\downarrow$
SimVP+gSTA-S	46.77M	26.60	77.32
SimVP+gSTA-L	524M	21.81	66.43
Ours (STEAM-Trans)	220M	21.19	64.34

ation yields improved prediction accuracy, concomitantly enhancing training and inference efficiency. This research contributes to the understanding of feature recycling’s impact on video prediction models, highlighting its potential for optimizing performance and streamlining computational processes.

3.5.2 Effect of Block Memory’s Updates

To investigate the efficacy of the memory update mechanism introduced in Section 3.3.1, we set $\alpha = 0$ in Equation 3.1. In doing so, we preclude the update of the block memory tensor (illustrated in Figure 3.3) during the generative process.

In our ablation experiments, we examine the consequences of disabling the memory update in the STEAM blocks during inference. The results of these experiments are presented in Table 3.3. Our findings indicate that the memory update step provides the model with a nuanced enhancement in performance, particularly in the context of long-sequence generation tasks. This suggests that the memory update mechanism plays a role in facilitating the model’s ability to capture and utilize temporal dependencies and contextual information.

3.5.3 Effect of the STEAM block

To investigate the impact of the STEAMon performance, we conducted an evaluation of a pre-trained Transformer (TransVar) model on the standard Moving-MNIST ($10 \rightarrow 10$) [67]. The results were compared to those of SimVP’s gSTA-S and gSTA-L variants. The sole difference between our model and the SimVP variants lies in the CNN blocks within the translator, with our model utilizing STEAMand SimVP employing gSTA. This comparison enables an analysis of the enhancements attributable to the STEAM.

Each model underwent training for 600 epochs, and the outcomes are presented in Table 3.4. From the results, we posit that the block memory and the external context tensor M , utilized in external attention, are components responsible for the improvement in performance. They facilitate the network’s extraction of global context. During pre-training, where iterative generation steps are absent, the block memory focuses on learning to retain global context as features, thereby enhancing the model’s generative capabilities.

3.5.4 Effect of Different Loss Functions

We analyze the consequences of excluding the three non-essential losses (Equations 3.6, 3.7, and 3.8) from the training process. The results, presented in Table 3.5, reveal that the target feature prediction loss (Equation 3.6) has the most significant influence on both Peak Signal-

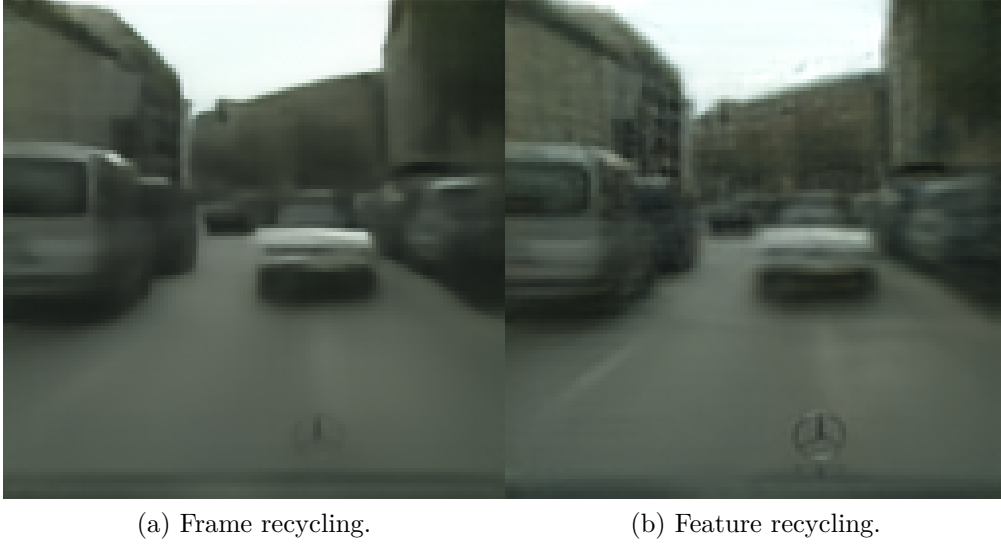


Figure 3.8 Comparison between the last frame ($t = 30$) generated using feature recycling and frame recycling. In the feature-recycled generation (right), the Mercedes logo on the car is more distinct and the building windows are clearer compared to the frame-recycled generation (left).

Table 3.5 A Loss Term Ablation Study. Columns one to three display the inclusion/exclusion status of various losses. To ablate the KL-divergence loss (Equation 3.8), we substitute the CVAE-Trans model with the STEAM-Trans model.

Equation 3.6	Equation 3.7	Equation 3.8	PSNR $\uparrow$	SSIM $\uparrow$
$\times$	$\checkmark$	$\checkmark$	25.72	0.583
$\checkmark$	$\times$	$\checkmark$	31.32	0.789
$\checkmark$	$\checkmark$	$\times$	31.19	0.784
$\checkmark$	$\checkmark$	$\checkmark$	31.34	0.790

to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) scores. This finding is consistent with expectations, given that the predicted feature has a direct impact on visual quality. The omission of this loss function consequently leads to a notable decline in image quality, as measured by these metrics.

3.6 Conclusion

In this work we introduce ViP-STEAM, a novel, lightweight, and purely convolutional neural network (CNN)-based approach to video prediction. Our key contribution lies in the development of the innovative ModelBlock structure, which serves as a repository for global context and features a memory update mechanism. This mechanism enables the model to write to and read from the repository, effectively capturing and utilizing global context to enhance prediction capabilities.

We demonstrate the efficacy of our approach by shifting the iterative generation process from the frame level to the feature level, thereby mitigating the compounding of errors that typically arises from repeated re-encoding of past images during autoregressive generation. This strategy significantly improves the accuracy and efficiency of video prediction.

Notably, our model achieves state-of-the-art (SOTA) results on multiple metrics across popular benchmarks, demonstrating its competitiveness even against diffusion and latent injection models. Our findings suggest that CNN-based approaches, despite being lightweight, can be highly effective in video prediction tasks and warrant further exploration. ModelName provides a robust baseline for future optimizations of purely CNN-based models, paving the way for advancements in this field.

CHAPTER 4 CTRL-V: HIGHER FIDELITY VIDEO GENERATION WITH BOUNDING-BOX CONTROLLED OBJECT MOTION

Video generation has garnered a lot of interest. In particular, video generation controlled by specific criteria is among the most popular research topics. By enabling precise control over aspects such as object placement, motion trajectories, and object type, controllable video generation empowers users to generate custom videos that meet particular requirements and creative visions. In this work, we explore an approach to control all these aspects in generated videos. Specifically, we propose a two-part bounding-box controlled video generation model. We introduce a bounding box predictor network that render pixel-level bounding boxes, and a video predictor that uses these bounding boxes as conditioning to create high-quality video. We perform experiments across three well-known autonomous vehicle video datasets: KITTI, Virtual-KITTI 2 and BDD100k.

In this collaborative work, my contributions are as follows:

- Formulation of the task.
- Design and implementation of various bounding box predictor models.
- Partial architecture of the model.
- Design of control input format and adapters.
- Partial implementation of input integration.
- Design of our own variant of Average Precision metrics.
- Running experiments and evaluations.

This work has been submitted to Conference on Robot Learning 2024. An arxiv version of the paper has been submitted to arxiv as well. Much of the content in the arxiv version overlaps with what is presented here.

4.1 Introduction

Recent breakthroughs in controllable image generation using text-to-image diffusion models [43, 78] have enabled the creation of highly realistic images based on conditioning inputs like sketches, depth maps, and segmentation maps. However, to better model real-world scenarios

for applications like autonomous vehicle navigation and safety, it’s essential to account for time.

Simulation plays a crucial role in training and testing autonomous vehicles, with most state-of-the-art systems relying on traditional computer graphics environments like CARLA [79]. However, experts envision a future where generative models can construct simulators, revolutionizing the way we train and test AVs. This vision is becoming a reality with commercial-grade neural sensor simulation approaches like UniSim [80].

Our work focuses on generating high-fidelity controllable video by developing a method that produces first-person vehicle videos conditioned on 2D and 3D bounding box predictions. Using a diffusion approach, our method predicts object motions and dynamics as sequences of bounding boxes and generates video accordingly. Our experiments demonstrate the ability to generate video that accurately follows desired bounding box motion conditioning, depicting precise object movements. Additionally, our method allows for supervision on object appearance and disappearance, including cars, pedestrians, and bikers.

To this end, we introduce **Ctrl-V**, a diffusion based bounding-box conditional video generation method that makes the following contributions and addresses multiple challenges associated with generating higher fidelity video from diffusion techniques:

1. **2D-Bounding-Box and 3D-Bounding-Box Conditioning:**

Our approach extends beyond merely managing motion with 2D bounding boxes; it permits 3D bounding boxes to control image generation, providing an even more detailed level of control. At the same time, our method remains compatible with widely used 2D bounding boxes.

2. **Bounding-box Motion Predictions with Diffusion:**

We propose a novel diffusion model based approach for predicting bounding box sequences in a video given the initial and final bounding box location.

3. **Uninitialized Object Generation:**

Current bounding-box controlled video generation models typically handle objects that are present either throughout the entire clip or that appear in the first frame and persist until the middle of the clip. New bounding boxes that emerge after the first frame are often disregarded or inadequately addressed, as noted by Wang et al. [1]. In contrast, our work trains a model to be responsive to all bounding boxes, regardless of whether they are present from the first frame or appear later in the clip. This ensures comprehensive handling of object appearances throughout the video sequence.

4.2 Preliminaries

4.2.1 Preliminaries

We begin here with an overview of the Stable Video Diffusion (SVD) model [41], due to its importance in our approach. SVD is a diffusion based image-to-video (I2V) model performed in latent embedding space [38]. Using an image $\mathbf{f}^{(0)}$ as initial condition, SVD is able to extend that single frame into a video $\mathbf{f} = [\mathbf{f}^{(0)}, \dots, \mathbf{f}^{(N)}]$ where N is the length of the sequence. Notably, SVD operates in latent space, where the diffusion and denoising process act upon the latents $\mathbf{z}$ of the video $\mathbf{f}$. Here, SVD employs an image encoder ($\mathcal{E}$) and an image decoder ($\mathcal{D}$) to translate each frame into and out of latent space: $\mathcal{D}(\mathcal{E}(\mathbf{f}^{(i)})) = \mathcal{D}(\mathbf{z}^{(i)}) \approx \mathbf{f}^{(i)}$. At each diffusion step, SVD progressively introduces noise into the latent representations. In this work, the amount of noise is dictated by Euler discrete noise scheduling method (EDM) introduced in Karras et al. [81]. A UNet based denoiser network within the SVD is used to predict this noise in order to recover the original latent representations. The UNet, $\mathbb{U}_\theta$, is parameterized as:

$$\mathbb{U}_\theta(\hat{\mathbf{z}}_t, \mathbf{z}_{\text{pad}}^{(0)}, \mathbf{c}^{(0)}, t), \quad (4.1)$$

- $\hat{\mathbf{z}}_t \in \mathbb{R}^{N \times C' \times H' \times W'}$: latent representation of frames corrupted by noise at noise level t .
- $\mathbf{z}^{(0)} \in \mathbb{R}^{1 \times C' \times H' \times W'}$: latent representation of the initial frame.
- $\mathbf{z}_{\text{pad}}^{(0)} \in \mathbb{R}^{N \times C' \times H' \times W'}$: Padded $\mathbf{z}^{(0)}$ by repeating itself along the first dimension N times.
- $\mathbf{c}^{(0)}$: CLIP encoding [82] of the initial frame.

The full denoiser network, $\mathbb{D}_\theta$, with an EDM noise scheduler, is formulated as

$$\mathbb{D}_\theta(\mathbf{z}; \mathbf{c}^{(0)}, \sigma_t) = \lambda_{\text{skip}}(\sigma_t)\mathbf{z} + \lambda_{\text{out}}(\sigma_t)\mathbb{U}_\theta(\lambda_{\text{in}}(\sigma_t)\mathbf{z}, \mathbf{z}_{\text{pad}}^{(0)}, \mathbf{c}^{(0)}; \lambda_{\text{noise}}(\sigma_t)) \quad (4.2)$$

Here λ_{skip} , λ_{out} , λ_{in} and λ_{noise} denote scaling functions, while σ_t represents the computed noise at level t . The precise mathematical definitions of these terms are detailed in Appendix A. Note that 3D UNet $\mathbb{U}_\theta$ in Equation 4.1, is a re-parameterized version of the one in Equation 4.2 [83]. The scaling terms are absorbed and the inputs are simplified for clarity. In the following sections, we follow the re-parameterized version in Equation 4.1 when referring to the UNets in our model.

4.3 Related Work

4.3.1 Video latent diffusion models

Video latent diffusion models (VLDMs) extend latent image diffusion techniques like those introduced in the work by Rombach et al. [84] to video generation. In early VLDMs works [38, 41, 85, 86, 87], most controllability comes from text-prompts or image-prompts as they were the main conditioning methods that can result in temporally consistent frame generation. Other modes of precise control often results in incoherent generations as these models struggle with complex scenes.

4.3.2 Conditional Video Diffusion

In order to have more control on the generations, more explicit conditioning are introduced in video diffusion work such as VideoCompose [88], Dreamix [89], Pix2Video [90], and DreamPose [91]. These work propose different adapters specific to the modes of control of interest. They act on top of VLDMs to transform conditioning input in order to achieve frame-level control. A particularly interesting branch of controllable generation is **ControlNet Adapted Video Diffusion**. Through incorporating ControlNet [43] adapters into the VLDM frameworks, these models are able to exert precise regional or pixel-level control in video generation. Some examples of these models are Control-A-Video [92], Video ControlNet [93, 94], ControlVideo [95], and ReVideo [96]. These work all show that ControlNet adapters can be used effectively with various types of conditioning; they are simple to train, and are able to achieve desired controllable effect with high precision and accuracy in the generated video.

4.3.3 Modes of Control

Controllable video generation research have been exploring many different modes of control. For instance, Control-A-Video [92] combines a pre-trained text-to-video model with ControlNet to manipulate videos using similar visual information. ControlVideo [95] utilizes a training-free strategy that employs pre-trained image LDMs and ControlNets to generate videos based on canny edge and depth maps. Video ControlNets [93, 94] uses optical flow as control, while ReVideo [96] generates video controlled by trajectories. DreamPose [91] injects pose sequence information into the initial noise. VideoComposer [88] uses a combination of depth, sketch, masks, and motion vectors as control.

A notable problem with many of the existing control modes, such as optical flow maps, depth maps, edge maps, is that they are full blown image sequences that are costly both

in terms of time and complexity. Furthermore, they often are difficult to change locally – imaging changing an object’s position in a depth map, it requires editing both areas around the object’s original position and its new position, not to mention the new depth has to be consistent with the surrounding area. Among these modes of control, bounding box emerge as one that has the flexibility that overcomes these issues. They are easily customizable and can be edited into different shape, size, locations and classes efficiently.

4.3.4 Bounding Box Based Models

Boximator[1] and **TrackDiffusion** [97] are two of the most relevant bounding box controlled video generation models. In order to incorporate bounding boxes information, both work introduces 3D U-Net blocks and a set of novel attention layers to process those information into the input of the U-Nets. In addition to bounding box coordinates, object IDs are also used as part of the input to these attention layers. This set of bounding-box input is transformed using a Fourier embedder [98] before being processed by an encoder of multi-layer perceptrons. In contrast, our proposed model make use of ControlNet and we do not need to train additional encoding layers or utilize Fourier embedder to process the bounding-box input. Furthermore, in order to ensure that the generated video adheres to bounding-box inputs, Boximator proposes a self-tracking technique, later adopted by TrackDiffusion. This technique require to learn an additional object tracking task along with video generation; it mandates a two-stage training process where the model first learn video generation with target bounding boxes in frames, and later another one without the bounding boxes. The model’s performance markedly declines without this tracking technique. On the other hand, our model achieves is able to closely follow the given bounding box conditions without any additional stages of training.

4.3.5 Vehicle Oriented Generative Models

A particular work from autonomous driving domain that is related to controllable video genertion is DriveDreamer [99]. In its action-based video simulation model, bounding boxes are also used in order to generated a series of driving actions along with a video rendering of the scene.

Fourier embeddings [98] and CLIP embeddings [82] are used for bounding-box information encoding and categorization. This work does not directly fit in our problem setting, but it is an example of using bounding boxes as part of the control scheme.

4.4 Method

4.4.1 Overview of our Method: Ctrl-V

Our controllable video generation method is illustrated in Figure 4.1. It consists of two sequential steps:

1. First, we generate bounding box frames using our diffusion based bounding box predictor network, the **BBox Predictor**, which is shown on the left side of Figure 4.1. These frames contain only bounding boxes. They make up a video of moving (or stationary) bounding boxes and it serve as the "skeleton" for the generated video.
2. Then, we generate a video using our video generator network, **Box2Video**, shown on the right side of Figure 4.1, where the bounding boxes frames act as the control signal. The bounding boxes in each frame determine the objects generated in the corresponding frames of the video.

BBox Predictor and Box2Video each utilizes a modified SVD backbone – illustrated by the SVD backbone in Figure 4.1. These backbones are adapted to their respective generation tasks. Details of each model are presented in their individual sections: BBox Predictor – Section 4.4.2 and Box2Video – Section 4.4.3.

4.4.2 Ctrl-V: BBox Predictor

The BBox Predictor shown on the left in Figure 4.1 aims to predict object bounding boxes across all video frames using an SVD backbone. The four inputs to the model are $\hat{\mathbf{b}}_t$, $\mathbf{b}^{(0)}$, $\mathbf{b}^{(N-1)}$, $\mathbf{z}^{(0)}$, where: $\hat{\mathbf{b}}_t$ is the encoded "video" of bounding boxes with t levels of noise added; $\mathbf{b}^{(0)}$ is the encoded initial bounding box frame(s); $\mathbf{b}^{(N-1)}$ is the encoded final bounding box frame; $\mathbf{z}^{(0)}$ is the encoded initial video frame. During training, the model learns to predict the noise added in $\hat{\mathbf{b}}_t$ according to the EDM noise scheduler. The model recovers the original $\mathbf{b}$ from its noisy version $\hat{\mathbf{b}}_t$ by calculating the noise with UNet outputs and eliminating the noise through scaling functions. We opt to abstract this detail in the model diagram for readability.

In practice, the four inputs are transformed and concatenated into a vector format accepted by the UNet adapter within the SVD backbone. Specifically, as shown in Figure 4.1, $\mathbf{z}^{(0)} \in \mathbb{R}^{1 \times C' \times H' \times W'}$ is replicated along the first dimension, and its front and end (in the first dimension) are replaced by $\mathbf{b}^{(0)}$, $\mathbf{b}^{(N-1)}$ respectively. This forms $\mathbf{z}_{\text{pad}}^{(0)} = \text{concat}(\mathbf{b}^{(0)}, \mathbf{z}^{(0)}, \dots, \mathbf{z}^{(0)}, \mathbf{b}^{(N-1)}) \in \mathbb{R}^{N \times C' \times H' \times W'}$. The noise-added encoding of bounding box video $\hat{\mathbf{b}}_t$ is then concatenated with

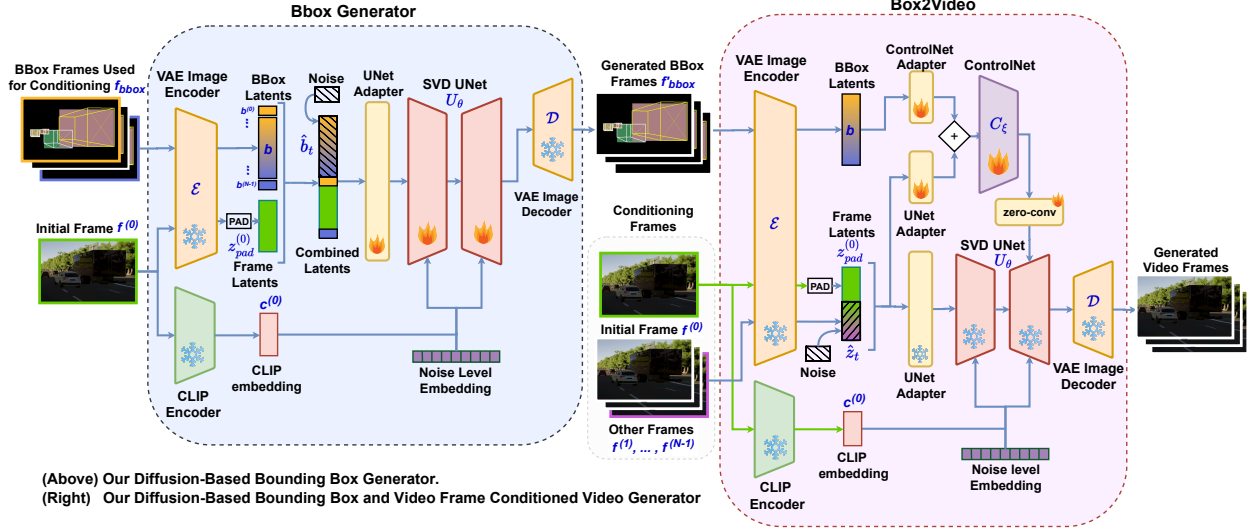


Figure 4.1 The diagram illustrates two components of **Ctrl-V**: (left) the **BBox Predictor** and (right) **Box2Video**. For both models, we use a **frozen, off-the-shelf VAE** to encode images into latent space ($\mathcal{E}$) and decode them back into pixel space ($\mathcal{D}$). During training, (1) the **BBox Predictor** (Sec. 4.4.2) learns to denoise the noisy bounding box frame latents $\hat{\mathbf{b}}_t$, conditioned on the first ($\mathbf{b}^{(0)}$) and last ($\mathbf{b}^{(N-1)}$) bounding box frame latents and the padded initial frame latent $\mathbf{z}_{pad}^{(0)}$ and (2) the **Box2Video** (Sec. 4.4.3) denoises the target frame latents $\hat{\mathbf{z}}_t$ by conditioning on the initial frame’s latent $\mathbf{z}_{pad}^{(0)}$ (input to the SVD UNet) and the bounding box frame latents $\mathbf{b}$ (input to the ControlNet).

$\mathbf{z}_{pad}^{(0)}$ to form the final input to the UNet adapter. The network incorporates additional conditioning inputs, including a CLIP-encoded embedding of the initial frame $\mathbf{c}^{(0)}$ and a noise-level embedding t . These embeddings are individually integrated into every sub-block of the U-Net through a self-attention mechanism.

Representing Bounding Boxes in Pixel Space

An important element of Ctrl-V is our design choice of rendering bounding boxes in pixel space. The manner in which bounding box information is provided as a control signal to the video generator is important. For example, prior work such as Boximator [1] represents bounding boxes as a Fourier transformed concatenated vector of their raw coordinates, ID and other information. In contrast, in our work we choose to render bounding boxes into frames while maintaining minimal loss of meta information. Importantly, we also encode information such as track ID, object type, and orientation for each bounding box using a combination of visual attributes, including border color, fill color, and markings. Specifically, the *track ID* represents a unique identifier for each tracked object across frames, the *object*

type specifies the category of the object (e.g., car, pedestrian), and the *orientation* indicates the direction the object is facing. Crucially, our approach allows us to leverage the highly effective ControlNet approach to provide pixel-level guidance to influence diffusion generated imagery.

4.4.3 Ctrl-V: Box2Video

Box2Video is shown on the right in Figure 4.1 and it aims to generate high-fidelity videos controlled by bounding box frames, such as those generated by the BBox Predictor network. Box2Video consists of an SVD backbone for video generation, and an adapted ControlNet module to process the bounding box control signal. ControlNet is a widely used network for controlling image generation. In this work, we modify ControlNet and adapt it to the video diffusion framework (as shown on the right in Figure 4.1). This architecture allows us to train Box2Video in a single stage without the need for additional optimization criteria, in contrast to previous work such as Boximator and TrackDiffusion [1, 2], which require multi-stage learning with extra criteria to train their models.

The SVD component takes two inputs: $\mathbf{z}^{(0)}$ and $\hat{\mathbf{z}}_t$. Here, $\mathbf{z}^{(0)}$ is the encoded initial video frame and $\hat{\mathbf{z}}_t$ is the encoded full video with t levels of noise added to it. As shown in Figure 4.1, we process these inputs by padding $\mathbf{z}^{(0)}$ by repeating it along the first dimension before concatenating it with $\hat{\mathbf{z}}_t$ to create the final input to the UNet adapter of the SVD. The same input is also sent to the ControlNet module through its own UNet adapter layers. Additionally, ControlNet also receives the encoded bounding box frames, $\mathbf{b}$, as input, through ControlNet adapter layers. Both of these transformed input is then added together before processed by the ControlNet module. The output signal of the ControlNet module then goes through a zero-convolution before being sent to the SVD UNet decoder layers through residual paths as control signal. During training, the weights of the SVD model (θ) are frozen, while only the weights in the ControlNet (ξ) are updated.

4.5 Experiments and Ablation Studies

For experiments on each dataset, all experiments are trained on the training sets and evaluated on the testing sets. All evaluation are done by randomly selecting 200 frames from the testing set of each dataset as the initial frames.

Furthermore, each bounding-box predictor discussed in the following section requires conditioning on ONE initial frame, ONE or THREE initial bounding-box frames and ONE final bounding-box/trajectory frame.

4.5.1 Datasets

We train and evaluate our models across three datasets: KITTI [100], Virtual KITTI 2 (vKITTI) [101], and the Berkeley Driving Dataset (BDD) [102] with Multi-object Tracking labels (MOT2020).

KITTI contains 22 real-world driving clips with 3D object labelling. The downloadable content of the official KITTI dataset does not include bounding-box label files for the testing set. As a result, we manually partition the dataset’s training folder into two segments for our experiments. The training split encompasses samples ‘0000’ to ‘0018’, while the testing split consists of samples ‘0019’ and ‘0020’.

Virtual-KITTI 2 (vKITTI) consists of 5 virtual simulated driving scenes. There is no official or unofficial specification of the train/test split. During training, we exclude ‘Scene20’, and the models were trained on the remaining four scenes.

BDD100k is a large-scale real-world driving dataset. It contains 1603 instances of driving clips, all of which are 2D-labeled. We use the official train/validation split.

For all clips, we resize the frame dimension to 312×520 , and use process clips to have 25 frames, more than the number used by prior work, which results in a more difficult video generation task.

4.5.2 Training Configurations

Our pretrained SVD models are adapted from HuggingFace’s diffuser library [103]. Specifically, we utilize the ‘stable-video-diffusion-img2vid-xt’ SVD variant in this project.

4.5.3 Metrics

Generation Quality Metrics

We use four widely used metrics to evaluate generation quality. PSNR, SSIM, LPIPS, and FVD. It is important to know all but FVD are designed for image quality evaluation, and they have well known shortcomings. Here we explain each metric used and their known problems.

PSNR, or Peak signal-to-noise ratio, calculates the ratio between the maximum value of a signal and the noise that affects the fidelity of its representation. It is closely related to Mean Squared Error (MSE) loss. It can be in fact calculated by subtracting the log of MSE from a constant. It is long been used as a metric for image reconstruction quality for its

ease of use and mathematical properties. However, PSNR suffers multiple issues as a metric for image generation tasks. Its insensitiveness to various distortion such as blurring, pepper noise, and mean-shifting means that image pairs that look very different to human can have very similar PSNR scores [104, 105]. For original and reconstructed images $\mathbf{I}$ and $\mathbf{K}$ of size $M \times N$

$$\text{PSNR} = 10 \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right) \quad (4.3)$$

$$\text{MSE} = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N (\mathbf{I}_{i,j} - \mathbf{K}_{i,j})^2 \quad (4.4)$$

MAX is the maximum possible pixel value of the image. MSE is the Mean Squared Error.

SSIM, or Structural similarity index measure, is a metric that aims to evaluate similarity in "structures" of image pairs – as opposed to absolute errors as in the cases of MSE and PSNR. Because the human visual system is more sensitive to structural distortions, this metric is more closely aligned with human evaluation compared to PSNR. However, some visually apparent distortions are not captured by the SSIM. Things like changing hue and brightness do not seem to affect SSIM much [106] and Sharif et al. [107] presents more examples of unpredictable ssim behaviours when presented with different distortions.

$$\text{SSIM}(\mathbf{I}, \mathbf{K}) = \frac{(2\boldsymbol{\mu}_I\boldsymbol{\mu}_K + C_1)(2\boldsymbol{\sigma}_{IK} + C_2)}{(\boldsymbol{\mu}_I^2 + \boldsymbol{\mu}_K^2 + C_1)(\boldsymbol{\sigma}_I^2 + \boldsymbol{\sigma}_K^2 + C_2)} \quad (4.5)$$

$\boldsymbol{\mu}_I$ and $\boldsymbol{\mu}_K$ are the mean intensities of $\mathbf{I}$ and $\mathbf{K}$. $\boldsymbol{\sigma}_I^2$ and $\boldsymbol{\sigma}_K^2$ are the variances of $\mathbf{I}$ and $\mathbf{K}$. $\boldsymbol{\sigma}_{IK}$ is the covariance of $\mathbf{I}$ and $\mathbf{K}$. $C_1 = (k_1L)^2$ and $C_2 = (k_2L)^2$ are constants to stabilize the division, with L being the dynamic range of the pixel values (e.g., 255 for 8-bit images), and k_1 and k_2 being small constants (e.g., $k_1 = 0.01$ and $k_2 = 0.03$).

LPIPS, or Learned Perceptual Image Patch Similarity [108], is a neural network based approach to accessing image similiarity. It does so by computing distance between some reprentation of image patches. These representations are obtained by running image patches through some pre-defined neural network. LPIPS has been shown to math human perception well.

$$\text{LPIPS}(\mathbf{I}, \mathbf{K}) = \sum_l \frac{1}{H_l W_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} \|\phi_l(\mathbf{I})_{hw} - \phi_l(\mathbf{K})_{hw}\|_2^2 \quad (4.6)$$

ϕ_l represents the feature activations at layer l of the network. H_l and W_l are the height and width of the feature map at layer l . $\|\cdot\|_2$ denotes the Euclidean distance (L2 norm).

FVD, or Frechet Video Distance [109], is a metric for generative video models. It is based on the Fréchet inception distance (FID) for images. Unlike the 3 other metrics that focuses on image frames and do not consider their temporal relationships, FVD is specifically designed for videos. It takes into consideration the visual quality, temporal coherence, and diversity of samples of videos generated by the model under evaluation. The large scale human study carried out by the authors suggest FVD consistently agrees with human evaluation of videos.

$$\text{FVD} = \|\boldsymbol{\mu}_r - \boldsymbol{\mu}_g\|_2^2 + \text{Tr}(\boldsymbol{\Sigma}_r + \boldsymbol{\Sigma}_g - 2(\boldsymbol{\Sigma}_r \boldsymbol{\Sigma}_g)^{1/2}) \quad (4.7)$$

$\boldsymbol{\mu}_r$ and $\boldsymbol{\mu}_g$ are the means of the real and generated video feature distributions, respectively. $\boldsymbol{\Sigma}_r$ and $\boldsymbol{\Sigma}_g$ are the covariances of the real and generated video feature distributions, respectively. Tr denotes the trace of a matrix.

Motion Control Metrics: Mean Average Precision

Intersection over Union (IoU): IoU is a measure used to determine whether a predicted bounding box matches a ground truth bounding box. It is calculated as the area of overlap between the predicted and ground truth bounding boxes divided by the area of their union. Different threshold set on IoU results in different precision and recall values, which are in turn used to calculate Average Precision scores.

Mean Average Precision (mAP) is used for evaluating object detection models. It integrates precision and recall over different thresholds to give a single score that reflects the model’s performance across various confidence levels.

Average precision (AP) is the area under the recall-precision curve; it takes into account the trade-off between precision and recall at different confidence thresholds. It serves as a measure to evaluate how well the predicted or generated bounding boxes correspond to the actual ground-truth labels.

We examine previously used custom mAP metrics. We run YOLOv8 and YOLOv8+BoT-SORT on the three datasets using detection and tracking experiments. We generate Track-mAP scores used by trackDiffusion [2] to evaluate the alignment score between the ground-truth frames detections and actual ground-truth labels. The results show a low recall value, which is caused by large amount of missing positive detections. This issue is more apparent when BoT-SORT is used for tracking. To be consistent, we decide to run detections from the ground-truth frames as the labels for our mAP calculations.

We use YOLOv8 detection model on both the generated and ground-truth videos. The

confidence cutoff fixed at 0.6. The bounding boxes from both sets of videos are then used for evaluations. Specific YOLOv8 configuration are detailed in Appendix A.

mAP is calculated by averaging the AP scores at different IoU thresholds following the MS COCO protocol [110]: we compute 10 AP scores at 10 different IoU thresholds from 0.5 to 0.95 in 0.05 increment. AP_τ denotes the AP score computed at IoU threshold $\tau\%$.

4.5.4 Generation Quality

Table 4.1 Evaluation on generation quality. Generated videos are 25 frames with a resolution of 312×520 .

	Pipeline	# Cond. BBox	FVD↓	LPIPS↓	SSIM↑	PSNR↑
KITTI	Stable Video Diffusion Baseline [41]	None	552.7	0.3504	0.4030	13.01
	BBox Predictor + Box2Video	1-to-1	467.7	0.3416	0.3241	13.21
	BBox Predictor + Box2Video	3-to-1	422.2	0.2981	0.4277	13.85
	Teacher-forced Box2Video	All	435.6	0.2963	0.4394	14.10
vKITTI	Stable Video Diffusion Baseline [41]	None	331.0	0.2852	0.5540	16.60
	BBox Predictor + Box2Video	3-to-1	341.4	0.2645	0.5841	17.60
	Teacher-forced Box2Video	All	313.3	0.2372	0.6203	18.41
BDD	Stable Video Diffusion Baseline [41]	None	409.0	0.3454	0.5379	16.99
	BBox Predictor + Box2Video	1-to-1	412.8	0.2967	0.5470	17.52
	BBox Predictor + Box2Video	3-to-1	373.1	0.3071	0.5407	17.37
	BBox Predictor ^{Traj} + Box2Video	3-to-1 ^{Traj}	375.8	0.2973	0.5481	17.55
	Teacher-forced Box2Video	All	348.9	0.2926	0.5836	18.39

In addition to our BBox Predictor + Box2Video model, we train two additional baseline models for comparison, namely fine-tuned SVD baselines, and teacher-forced Box2Video generation model. Fine-tuned SVD baseline is a standard pretrained SVD model finetuned to generate video from initial frame(s), while the teacher-forced Box2Video generation model take the initial frame as well as all of the ground truth bounding box frames as input to generate videos. In comparison, our BBox Predictor + Box2Video model takes the initial frame and the initial bounding frame as input.

We evaluate generation quality using the four metrics of PSNR, SSIM, LPIPS, and FVD introduced in Section 4.5.3. In addition to using just 1 initial frames, we also experiment with 3 initial frames. We also experiment with having the last frame as the trajectory instead of full bounding boxes. These variants are indicated by the "# Cond. BBox" column, where m-to-n indicates m initial frames used and n final frames used, and "traj" superscript indicate the use of midpoint trajectory frames instead of bounding box frames.

The evaluated results are reported in Table 4.1 and visualizations are shown in Figure 4.2. These results indicate that the generation quality improves as we condition on more ground-truth bounding-box frames.

More generation results are available in Appendix A.



(a) SVD's generation at frame 13



(b) ControlNet's generation at frame 13



(c) SVD's generation at frame 13



(d) ControlNet's generation at frame 13

Figure 4.2 Comparison of an urban scene generation between the baseline Stable Video Diffusion (SVD) model and the teacher-forced Box2Video model.

4.5.5 BBox Predictor: Quantitative Evaluation

Training Details

We train various types of bounding-box predictors in this work:

- 3D bounding-box predictor (3-to-1)
- 2D bounding-box predictor (3-to-1)
- 2D bounding-box-trajectory predictor (3-to-1^{Traj})

Table 4.2 All experiments reported in this table are conditioned on 3 initial bounding-box frames and 1 final bounding-box or trajectory frame. The first three columns show evaluations on the entire generated bounding-box sequence, measuring the alignment scores between our generated bounding box predictions and ground-truth labels. The last three columns focus on testing the auto-encoding capability of the network, evaluating only the first and last frames of the generated sequence. If a dataset’s name is labelled with “Traj” superscript, it indicates that in the experiment reported in that row, the last bounding-box frame is replaced with its trajectory frame.

	# Cond. Frames	maskIoU $\uparrow$	maskP $\uparrow$	maskR $\uparrow$	maskIoU $\uparrow$ (first+last)	maskP $\uparrow$ (first+last)	maskR $\uparrow$ (first+last)
KITTI	1-to-1	.629 $\pm$.212	.763 $\pm$.188	.758 $\pm$.176	.986 $\pm$.012	.994 $\pm$.008	.992 $\pm$.009
	3-to-1	.795 $\pm$.112	.884 $\pm$.078	.881 $\pm$.082	.986 $\pm$.010	.992 $\pm$.007	.994 $\pm$.005
vKITTI	1-to-1			.769 $\pm$.212			
	3-to-1	.767 $\pm$.131	.853 $\pm$.078	.881 $\pm$.126	.944 $\pm$.039	.948 $\pm$.036	.996 $\pm$.006
BDD	1-to-1	.587 $\pm$.214	.712 $\pm$.194	.747 $\pm$.187	.954 $\pm$.047	.955 $\pm$.047	.999 $\pm$.002
	3-to-1	.647 $\pm$.176	.783 $\pm$.156	.784 $\pm$.150	.955 $\pm$.043	.955 $\pm$.042	.997 $\pm$.001
	3-to-1 ^{Traj}	.606 $\pm$.201	.773 $\pm$.162	.722 $\pm$.189	.798 $\pm$.155	.886 $\pm$.137	.894 $\pm$.118

The 3D bounding-box predictor predict 3D bounding boxes, while the 2D predictor predict 2D bounding boxes. A 1-to-1 model predicts frames between the first and last bounding-box frames, while a 3-to-1 model predicts the intermediate frames from frame 4 to 2nd-last bounding-box frame. Additionally, there is a bounding-box-trajectory model where the last conditional frame is a trajectory frame instead of a bounding-box frame, an example of this trajectory frame is shown in Figure 4.3

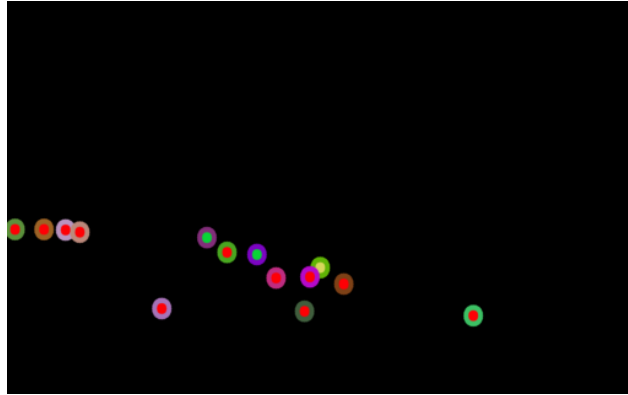


Figure 4.3 Example visualization of a trajectory frame, where the bounding boxes are replaced by their mid-point traces.

For the bounding-box predictors trained on KITTI or vKITTI datasets, each model undergoes 5 epochs of training. The 2D bounding-box predictor (1-to-1) is trained on BDD100K

for 33,000 iterations. Additionally, the 2D bounding-box (3-to-1) and 2D bounding-box-trajectory (3-to-1) models start their fine-tuning process for an additional 15,000 iterations from where the 2D bounding-box predictor (1-to-1)’s training ends. All of these prediction models are trained on a single A100 GPU with a batch size of 1. We also set gradient accumulation steps to 5 and employed AdamW [111] as our optimizer.

The fine-tuning and training procedures for the baseline Stable Diffusion and the ControlNet models are almost the same as of the bounding-box predictor model. The only difference is that we trained our baseline and ControlNet for 48,000 iterations on BDD100K.

Evaluation

To evaluate the quality of our bounding-box predictions, we compare bounding box masks frames for both the ground-truth and predicted bounding-box frames. The bounding box masks are generated by converting the bounding-box frames into binary masks with each pixel either being part of a bounding box or not.

For each ground truth bounding box frames, we simply take the sum across across channel, and assign 1 to a pixel that has a sum larger than 0.

For each predicted bounding box frame, the procedure is more complicated due to the fact that the image rendered from latents generated by a network. Most pixel values are not exactly zero. We still take the sum across channels, but set a threshold at 50. All pixels with value less than 50 are set to 0; all other pixels are set to 1.

Comparing the generated mask frames with the ground truth mask frames, we define 3 metrics: mask Intersection over Union (maskIoU) scores, averaged mask Precision (maskP) scores, and averaged mask Recall (maskR) scores with the help of Figure 4.4:

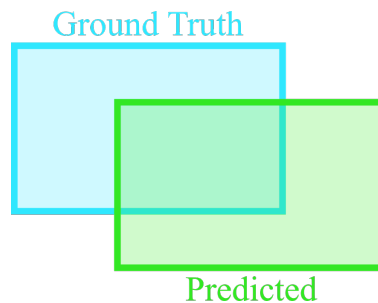


Figure 4.4 Figure to explain mask metric definitions. Let the region of ground truth bounding box be $\mathbf{T}$ and predicted bounding box region be $\mathbf{P}$

$$\text{maskIOU} = \frac{1}{N} \sum_{clip} \frac{\mathbf{T} \cap \mathbf{P}}{\mathbf{T} \cup \mathbf{P}} \quad (4.8)$$

$$\text{maskP} = \frac{1}{N} \sum_{clip} \frac{\mathbf{T} \cap \mathbf{P}}{\mathbf{P}} \quad (4.9)$$

$$\text{maskR} = \frac{1}{N} \sum_{clip} \frac{\mathbf{T} \cap \mathbf{P}}{\mathbf{T}} \quad (4.10)$$

$$(4.11)$$

N is the number of total clips. In essence, maskIoU measures the averaged ratio of the intersection area to the union area of the masks. The maskP is the precision of the predicted masks, and maskR is the recall of the masks.

In evaluation, we practice "best-of-K" method where we sample $K = 5$ bounding box sequences for each ground truth sequence, and select the sample with the highest maskIoU score for further evaluation. Results are reported in in Table 4.2, and sample generations are shown in Figure A.6 As shown in the results, as expected, the model shows a high score on the first and last frames since they are used as part of the conditioning. The score drops when we take into account the intermediate frames. A main factor of this is that the motion of object is not deterministic and in turn, BBox Predictor does not learn a deterministic set of bounding box movements. Rather, it learns a distribution of possible bounding box sequences. The aim of the BBox Predictor is to predict a temporally consistent set of bounding box sequences and this is not reflected in the commonly used bounding box metrics. Regardless of the scoring, we can see from Figure A.6 that the generated video from Box2Video closely adheres to the bounding boxes generated by the BBox Predictor.

4.5.6 Box2Video: Motion Control Evaluation

In this section, we present results on how closely the video generation follows the conditioning bounding boxes. We assess the fidelity of our Box2Videogenerations by analyzing their consistency with ground-truth bounding box conditions. To measure the accuracy of object locations, we employ the variant of mAP metric introduced in Section 4.5.3, tailored to accommodate the complexities of autonomous driving scenarios.

It is important to note that Average Precision scores are implemented slightly differently across models. For instance, Boximator [1] only accounts for bounding boxes of objects that persist throughout the entire video and disregard any other bounding boxes for their calculation. Meanwhile, TrackDiffusion [2] introduce their own version of AP called TrackAP which make uses of bounding boxes' track IDs to ensure boxes being compared shares the

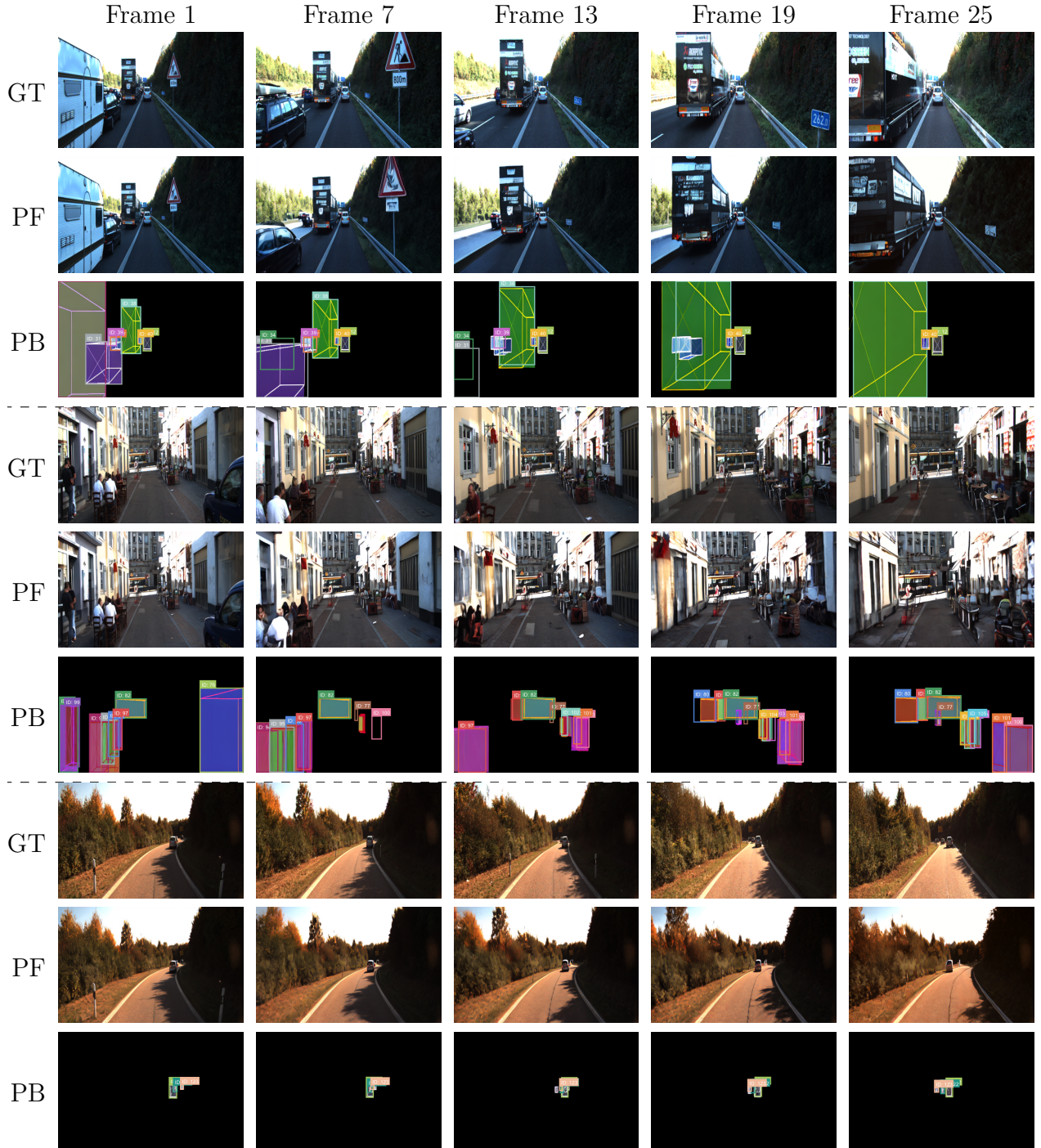


Figure 4.5 3D bounding-box predictions and motion-controlled video generations for various scenes on KITTI test-split.

same identity. However, the dataset used in TrackDiffusion contains low amount of objects in scenes, and TrackAP requires track ID to be available for all frames. Given these restrictions, we adapt mAP to our specific problem. Unlike previous studies, which focused on specific



Figure 4.6 Illustrations of the generations conditioned on ground truth 3D bounding boxes (2D for BDD) across various datasets. The 2D outlines of bounding boxes are overlaid on top.

aspects of object motion, our AP metric comprehensively evaluates all objects across every scene, including those that enter, exit, or overlap. We utilize YOLOv8 [112], a state-of-the-art object detection tool, to track objects in generated and ground-truth scenes. Then, we match objects based on spatial similarity, using intersection over union (IoU) scores to determine location similarity. Finally, we compute the average precision score following the MS COCO protocol [110].

Our results, presented in Table 4.3, demonstrate that our Box2Video model generate videos that closely adheres to the control bounding boxes. Our approach effectively controls object motions, generating videos that accurately align with ground-truth bounding box conditions.

The precision-recall curve for different IoU thresholds, used to calculate the mAP and AP scores are presented Figure 4.7

Precision-Recall Analysis Our precision-recall curves consistently surpass the diagonal line, indicating higher precision than recall. This suggests that when our detector identifies an object in a generated frame, it is likely to correspond to a detected object in the ground-truth frame. However, the lower recall rate indicates that objects may not always be detected in the same location in the ground-truth frame. We suspect this could be caused by one of three factors:

Object generation: The specific shapes or texture of generated objects may cause detected bounding box to vary slightly in location, this could affect recall.

Detection Tool: The object detection tool used directly impacts recall. With confidence

¹Boximator only used the last generated frame’s bounding boxes to compute average percision scores.

Table 4.3 Motion Control Evaluation. We include Boximator [1] and trackDiffusion [2]’s reported performances on other datasets to provide a comparative context, though they use slightly different evaluation metrics. Additionally, in general, video length is inversely proportional to detection rate and generation quality. Our longer generation compared to related works poses an additional challenge for our model.

Method	Dataset	Dataset Type	# Frames	mAP $\uparrow$	AP $_{50}\uparrow$	AP $_{75}\uparrow$	AP $_{90}\uparrow$
Ctrl-V	KITTI	Driving	25	0.547	0.712	0.601	0.327
	vKITTI	Driving-sim	25	0.599	0.776	0.667	0.356
	BDD	Driving	25	0.685	0.855	0.781	0.401
Boximator¹	MSR-VTT[113]	Web videos	16	0.365	0.521	0.384	-
	ActivityNet [114]	Human-action	16	0.394	0.607	0.409	-
	UCF-101 [115]	Human-action	16	0.212	0.343	0.205	-
TrackDiffusion	YTVIS [116]	YouTube videos	16	0.467	0.656	-	-
	UCF-101 [115]	Human-action	16	0.205	0.326	-	-

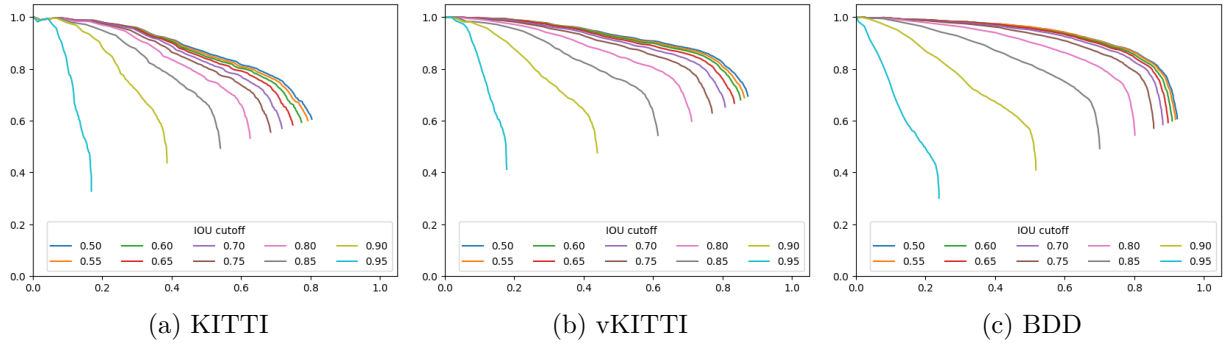


Figure 4.7 Precision-recall curves: the x-axis represents the recall rate, and the y-axis represents the precision rate.

threshold being a hyper parameter, more parameter search might be needed to find the best value to decrease the amount of false negatives which lowers recall.

Intrinsic Dataset Errors: Many examples in Kitti and vKitti dataset contain bounding boxes that do not have objects in them, which can lead to an increase in false negatives when generating videos. Moreover, the restriction of data processing and limitation of having a maximum number of bounding box (15 in our case) can result in missing bounding boxes, lowering recall.

As shown in Figure 4.7, the BDD experiment yield the best precision-recall curve. We suspect the significantly larger dataset size BDD can be a major factor in teaching the model to be more robust.

4.5.7 Model Creativity – Unconditioned Motion Generation

To examine the model’s ability to generate bounding box trajectories without full specification, we run generation without providing the model with the final bounding box positions by omitting the final bounding box frame. In other words, we give the BBox Predictor the freedom to generate plausible trajectories according to its creativity. We find that the model is able to generate plausible motions that are vastly different from the ground truth. In Figure 4.8 we showcase a couple sample generations, with the ground truth bounding box overlaid on top of the generation for comparison.

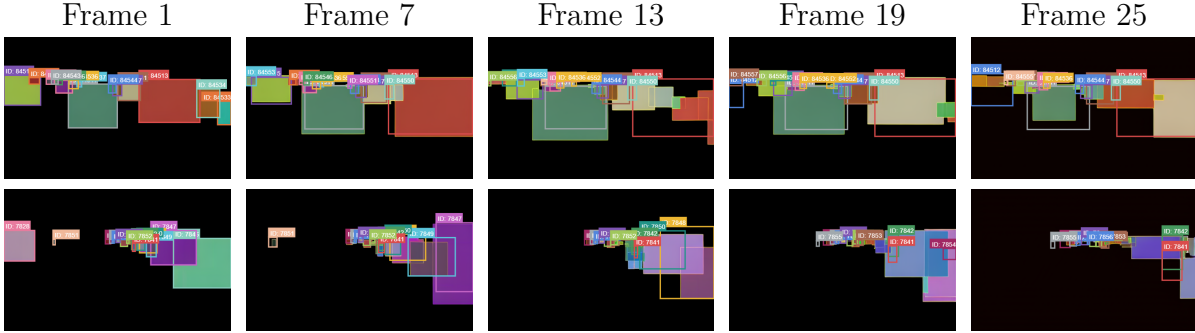


Figure 4.8 Bounding box trajectory generations are produced without conditioning on the last frame’s bounding box locations. The provided demos display generated trajectories with ground-truth bounding box trajectories overlaid for visual comparison.

We pass these generated trajectories to Box2Video to generate final videos and we find that our model is able to generate realistic videos with objects closely following the generated trajectories. Some examples are shown in Figure 4.9

4.6 Conclusion

In this work, we introduce a bounding box controlled video generation model, **Ctrl-V**, which consists of a bounding box predicting model BBox Predictor, and a conditional video generation model Box2Video. Through our evaluation, we demonstrate the effectiveness of the BBox Predictor model generating bounding box sequences that closely follow requirements for the first and last frames. We also show the quality of Box2Video’s generations, and how strictly they adhere to the control bounding boxes. On top of that, we introduce a novel procedure to generate bounding box renderings while retaining important 3D information. Our model can work with both 2D and 3D bounding boxes and handles uninitialized objects appearing in the middle of the videos. Lastly, we introduce a new variant of mAP metrics that allow comparison between high amount of untracked bounding boxes. Overall, Ctrl-V

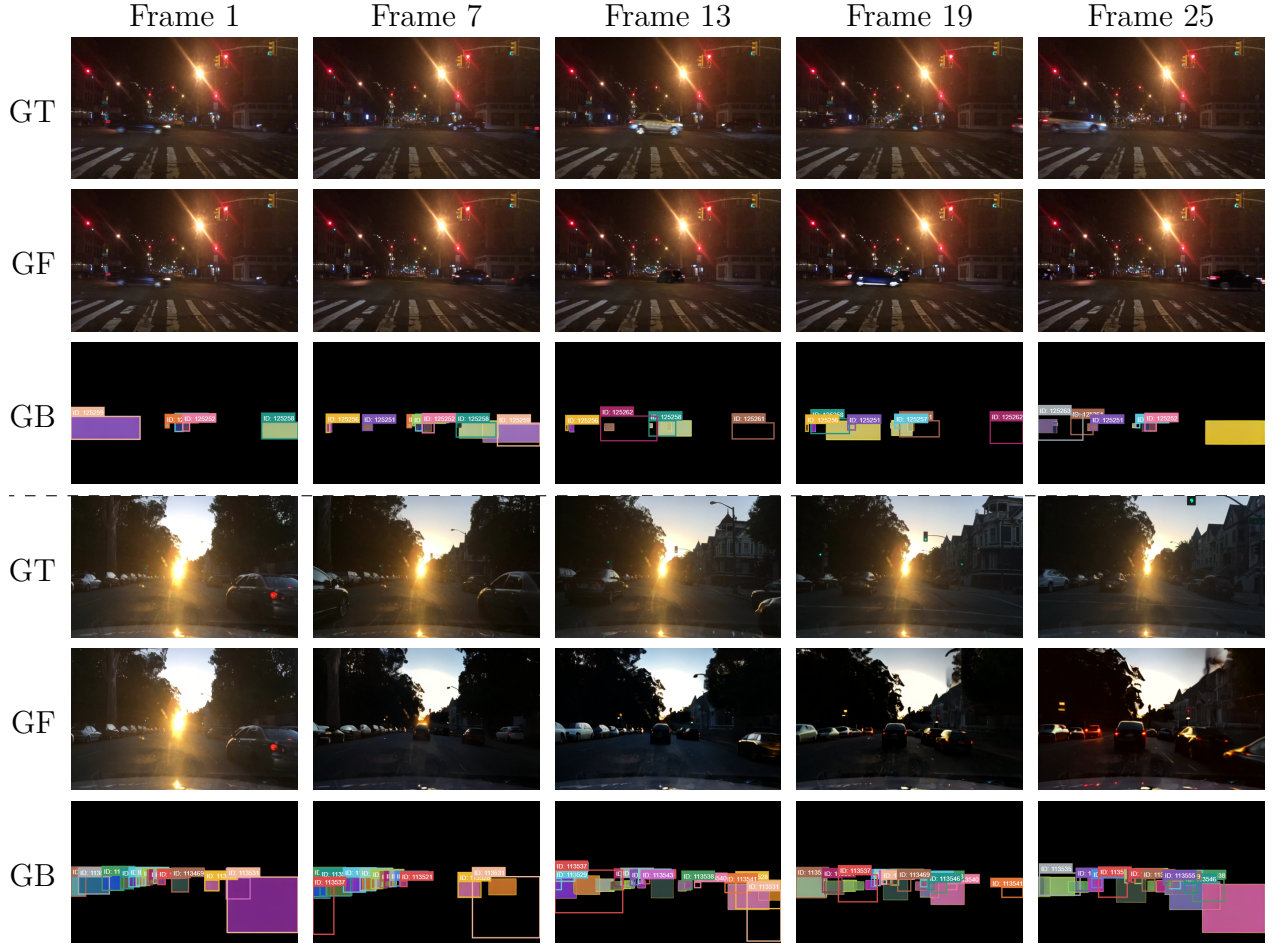


Figure 4.9 2D bounding box frame generations and motion-controlled video generations on BDD100K test split where no final bounding box frame conditioning was provided.

provides researchers with an efficient way generate video controlled by bounding boxes. It also provides a blueprint for other possible control modes to work with SVD models. With the aforementioned contributions, we believe Ctrl-V can be part of a solid foundation upon which future research in controllable video generation can be built.

Acknowledgments

We thank CIFAR for their support under the AI chairs program, NSERC for their support under the Discovery Grants program, and Samsung for supporting this work.

CHAPTER 5 DSTRUCT2DESIGN: DATA STRUCTURE DRIVEN GENERATIVE FLOOR PLAN DESIGN WITH TRANSFORMERS

Controllable and conditional generation, one of the most popular research topic currently. Within this field, text conditioned generative models for images represented as grids of pixels have yielded impressive results and received significant attention. In most works, language is used as a conditioning to a larger image generation model; however, there are many cases where it is beneficial to have a language based data structure as the input and main output which can then be transformed into pixel renderings. Moreover, in tasks that aim to output designs, it is crucial for the model to accomodate for numerical constraints and criteria that often arises in the design process. In this work we examine one of such tasks – the floorplan design generation problem. We introduce a large language model (LLM) data structure to data structure formulation of problem. We focus on incorporating different constraints in the form of language based data structure as means of controlling generation. Our formulation accepts partial or full design instructions and transforms it into a new data structure representing a floorplan. While LLMs are known to lacking when it comes to arithmetics, we find that with a data structure language, LLM can produce valid floorplans that respect the input constraints really well. Along with this finding, we examine different ways to improve the performance of LLMs for generative design, when quantitative constraints are only partially specified, but must be respected.

As the first author of this work, my contributions are as follows:

- Formulation of the research problem.
- Implementation of RPLAN dataset conversion.
- Partial implementation on ProcTHOR-10K dataset conversion.
- Design and implementation of prompt synthesis
- Implementation of fine-tuning tweaks.
- Design of evaluation metrics.
- Implementation of floorplan visualization.

This work has been submitted to the thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024). We plan to submit it onto arxiv as well. Much of the work presented here will overlap with the arxiv version.

5.1 Introduction

The floor design problem has garnered significant attention in recent years, driven by advancements in generative models and computing power. These developments have unlocked new possibilities for enhancing the design process. Current generative modeling-based tools have untapped potential to improve the floor plan generation process to incorporate more controllability.

Most previous approaches frame this problem as a computer vision task. Work such as FloorplanGAN [117], HouseGAN [118], HouseGAN++ [119], HouseDiffusion [120] focus on generating a two-dimensional top-down views of floorplan. Other work like AnyHome [121] and HoloDeck[122] aim to generate 3D renderings of floorplans. These methods are often conditioned on relational graphs of rooms. These graphs are known as bubble diagrams. As a design criteria, they present the desired connectivity between rooms. Some work attempts to add more numerical constraints as conditioning, but those approaches have limitations. For instance, FloorplanGAN [117], uses a table of rooms and their desired areas to condition a generative adversarial network (GAN), but it disregards the room connectivity information associated with bubble diagrams. Instead, the GAN proposes geometric configurations for rooms. Overall, the focus of prior work are on aesthetics, and it is apparent in the metrics used. Two of the three most common metrics in this problem are Diversity which is the FID score on images, and Realism, which is a human study score on how realistic the generated floorplans look.

In real-world scenarios, users often have diverse constraints and desires, other than ones that are aesthetics, that must be respected for a generated floor plan to be practical. In contrast to prior work, our approach focus on implementing numerical constraints, and creating new metrics that measures exactly how well the generated floorplans respect those specifications. We leverage pretrained Large Language Model’s (LLM) internal knowlegde, its ability to transform sequences and data structures to achieve this goal. Specifically, We transform user constraints and floorplans both into language based data structures, and use them to fine-tune a Llama 3 [21, 23] LLM. Constraints also include graph structures such as bubble diagram, in addition to numerical ones like room sizes, height, length. Floorplans are converted in a reversible process that specify the polygons that represent rooms of a floorplan. The reversibility of the data structure allows reconstruction of floorplan images from generated floorplan data-structure. Our approach enables the generation of floor plans that meet user requirements and constraints, revolutionizing the design process.

Our contributions are three-fold:

1. We contribute a unified floorplan dataset by merging existing popular datasets (RPLAN and ProcTHOR) and postprocessing them into a format which is amenable to language-based generative models which take as input and output structured text.
2. We design an array of metrics to benchmark the performance of the generated floorplans, in particular how well they respect the constraints set by the user.
3. We train and evaluate an LLM on our task setting to demonstrate the feasibility of our proposed problem formulation.

5.2 Related Work

5.2.1 Datasets

One of the most commonly used datasets for floorplan generation is RPLAN [123]. We provide more details of this dataset in Section 5.3.2. LIFULL [124] is an extremely large-scale dataset consisting of floorplans sourced from a Japanese realtor company. Floorplans are provided in raster format, however a subset of vectorized versions exist [125]. One common shortcoming is that there is a lack of well-annotated floorplan data (e.g. vectorised) which is amenable to training. This dataset is not widely used and we do not take data from it in our work. However, it is possible to apply our methods in this dataset in future works. Architext [126] aims to provide a text base representation for floorplan images. It addresses the issue of missing annotations by generating synthetic but diverse floorplans via a CAD script for Rhinoceros 3D; however, Rhinoceros 3D is not fully open source. In a similar vein, we leverage ProcTHOR [127], a fully open source simulator of buildings and floorplans. As it is one of the main dataset used in this work, we discuss it in more details in In Section 5.3.2.

5.2.2 Generative Models

Recent advancements in generative models have led to the development of various techniques for floorplan design. We provide an overview of approach of prominent methods in this section. We highlight their strengths and summarize their limitations.

House-GAN and House-GAN++ [128, 129] are convolutional graph-based networks that generate floorplans using GANs. While they demonstrate promising results, they only condition on bubble graphs, and do not support numerical constraints, limiting their flexibility.

FloorplanGAN [117] introduces a self-attention-based GAN that takes some numerical constraints like room center coordinates, desired areas, and room types as input. However, GAN

does not always respect the input constraints, and the method does not support partial specification of constraints or polygons. Moreover, the work does not explicitly evaluate how consistent the generated floorplans are with respect to the input quantitative criteria.

HouseDiffusion [120] is a diffusion-based method that predicts polygons for each room using a transformer architecture conditioned on a bubble diagram. Although it could potentially be conditioned on initial polygonal room specifications, its primary form of conditioning is via bubble diagrams.

ArchiText [126] leverages Large Language Models (LLMs) to generate floorplans from natural language descriptions. However, it does not support geometric inputs, limiting its versatility.

In summary, while these methods demonstrate progress in floorplan design, they have limitations in terms of conditioning flexibility, constraint respect, and geometric input support. Our approach aims to address these limitations by leveraging LLMs to transform user input into floor plans that meet their needs and constraints.

5.2.3 3D Scene Generation

Recent breakthroughs in full 3D scene generation have yielded remarkable results. AnyHome [121] and Holodeck [122] have demonstrated the ability to generate comprehensive 3D scenes. In addition to floorplans, these scenes can also contain objects like windows, doors, furniture. From a simple query like "a 1b1b apartment", these models are able to generate meaningful 3D renderings. While these methods showcase impressive capabilities, our approach focuses specifically on the floor plan aspect, offering unique features that set us apart.

Our method allows users to specify room dimensions and areas, as well as the total floor plan area, which is not supported by AnyHome and Holodeck. This added flexibility enables users to exert greater control over the generated floor plans. Moreover, our dataset is partially built upon ProcTHOR, a 3D-based floor plan simulator, which paves the way for potential future extensions to support the placement of furniture and other props. This could further enhance the versatility and realism of our generated floor plans.

While AnyHome and Holodeck have made significant strides in 3D scene generation, our approach offers a more specialized and customizable solution for floor plan design. By focusing on this specific aspect, we provide a more detailed and user-centric approach to generating floor plans, which can be valuable for various applications such as architecture, interior design, and real estate.

An overview of how our method compares to the others can be found in Table 5.1.

Table 5.1 **Comparison to related works.** Our method focuses on generation of floor plans as opposed to full 3D scenes, and it allows the user to specify individual room geometry (height, width, and/or area) and global apartment geometry in meters and sq. meters.

Method	Input	Output	Can specify Geometry?	Absolute Coordinates?
House-GAN, House-GAN++, HouseDiffusion	Bubble diagram (room adjacency)	Room masks	no	no
ArchiText	Natural Language	Room polygon coords	no	yes
FloorplanGAN	Fixed-Length Tensor	Room masks	yes	no
AnyHome, Holodeck	Natural Language	Furnished 3D models	no	yes
Ours	Structured Language	Room polygon coords	yes	yes

5.3 DStruct2Design Floorplan Dataset

In this research, we approach the problem of floor plan generation from a novel text-based data structure perspective. Our goal is to develop a unified method that enables users to input language commands, apply numerical constraints, and condition on graphs like bubble diagrams, which have been the traditional input in floor plan generation tasks.

To achieve this, it is important to develop an appropriate data representation for floor plans. Unlike previous work, which relied on 2D images or 3D scenes, we design a JSON-based data structure specifically tailored for our task. This new representation enables the efficient processing and generation of floor plans.

In our work, we create a new dataset by converting existing image and scene data into our JSON-based data structure. With this converted data representation, we can leverage the strengths of text-based input methods, combining the flexibility of natural language input with the precision of numerical constraints and graph-based conditioning.

By framing the problem in a text-based perspective, we aim to provide a more intuitive and user-friendly interface for floor plan generation. Our model enable users to impose quantitative constraints in their design requirements. And we remain compatible with the common formulation of using bubble diagram as conditioning. We are also able to generate floorplan images as output as in the common existing formulation by transforming our output floorplan datastructure into images.

5.3.1 Data Structure

Our JSON data structure contains essential numerical data, including the number of rooms, total area, and room types, for each floor plan. Each room is further detailed with room-specific fields, providing a comprehensive description. Notably, the location of each room is precisely defined by a set of vertices that form a polygon. During training, this forces the language model to predict exact coordinates rather than vague locations. This rigorous definition prevents the model from generating ambiguous room placements, resulting in more accurate and specific floor plan designs. The specific structure is presented in Table 5.2.

Overall, our designed data structure has several advantageous properties:

1. Employing exact values in our data structure gives us the ability to set numerical constraints in the input. More importantly, it facilitates the development of explicit metrics used to evaluate how closely the floorplan generation follow the input constraints.
2. Our data structure representation of floorplan contains sufficient information to be transformed into image renderings. This allow seamless integration with traditional visualization techniques, and makes our method compatible and easy to compare with other work in the same field.
3. Our JSON data structure boasts high extensibility, allowing for the seamless integration of additional information into the generation process. This flexibility enables the inclusion of various details, such as object placement and attributes within the floor plan, additional room features and specification, and custom user preferences and design requirements. This adaptability ensures that our approach can be easily expanded to accommodate various task extensions and applications in the future.

5.3.2 Datasets & Preparation

In this section, we discuss the two main datasets used in this work, and go into more details on their contents.

ProcTHOR-10k

ProcTHOR-10k [127] is a dataset of 12,000 procedural generated, fully interactive 3D houses designed for research in Embodied AI. The creation of the floorplans in this dataset was carried out through a four-step process by their authors. The first step involves defining room specifications (room specs), which determine the rooms in a house, their relative sizes,

Table 5.2 Processed Floorplan Structure. Each row is a field in the structure. Brackets "[]" indicate the value of the field is a list. Indented rows below indicate fields in the list.

"room_count"	▷ number of rooms
"total_area"	▷ total area in square units
"room_types" []	▷ list of string of room types present
"rooms" []	▷ list of individual room dictionary containing room specifics
"area"	▷ area of this room
"floor_polygon" []	▷ list of vertices that defines this room's layout
"x"	▷ x coordinate of the vertex
"y"	▷ y coordinate of the vertex
"is_regular"	▷ flag that indicates if the room's shape is rectangular
"height"	▷ the y-axis length of the rectangle bounding box enclosing the room
"width"	▷ the x-axis length of the rectangle bounding box enclosing the room
"id"	▷ unique id of this room
"room_type"	▷ the type of this room
"edges" []	▷ defines the connection between rooms for bubble gram generation

and how they are connected. This is achieved using a tree data structure where each node represents a room. Below the root node, each tree node is assigned a growth weight, which approximates the relative size of the node compared to its siblings. For PROCTOR-10K, 16 different room specs were utilized, each specifying between 1 to 10 rooms. In the second step, the size and shape of the house's interior boundaries are sampled. This process involves specifying the distribution of the house's dimensions per the room specs. The sampled interior boundary is cut, introducing randomness and variability. The method generates diverse house shapes by sampling the size and making these cuts while adhering to the specified constraints. In the third step, the algorithm proposed in [130] is employed to divide the interior boundary into rooms based on the room specs. This algorithm recursively subdivides the boundary according to the tree structure of the room specs. Growth weights influence the size of each subdivision, ensuring rooms are proportionally sized. In the last step, the authors connect the rooms by doors, but this connection is not represented in the two-dimensional space.

We clean and process the raw data from each house, focusing on the geometric properties of the rooms. We employ the shoelace formula to calculate areas and determined room dimensions based on their x and y coordinates. Additionally, we remove redundant points and apply rounding to the coordinates for consistency. Next, we categorize and count the types of rooms, and compute the total area for each house layout. The processed data is organized into the new JSON structure explained in Table 5.2.

RPLAN

RPLAN [123] is a manually collected dataset of 80,788 real world floor plans of buildings in Asia. Each floor plan in RPLAN is stored as a $256 \times 256 \times 4$ vector image. Channels 1 and 2 store interior and exterior boundary information; channel 3 contains room information where each pixel value denotes which room it belongs to; channel 4 has extra information to distinguish rooms with the same room type value in channel 3. To convert this 4 channel image into a JSON structure with well-defined room location, we first extract all pixel coordinates for all of the rooms. For each room, we locate the pixels that make up its perimeter. Then, by tracing the perimeter in one direction we are able to capture all of the vertices in an order that allows recreation of the room polygon. We take the room’s type and deduce all the other fields listed in Table 5.2 to complete the data structure. We convert 80,315 floorplans from RPLAN.

5.3.3 Bubble Diagrams

Bubble Diagrams: A Traditional Foundation for Floor Plan Design Previous work has often relied on bubble diagrams as a starting point for floor plan design, utilizing them to conceptualize the layout and spatial relationships between rooms. As illustrated in Figure 5.1 and Figure 5.2, bubble diagrams represent rooms as nodes and their connections as edges, providing guidance and constraints for the design process. Formally, a bubble diagram is defined as a graph $\mathcal{G} = (N, E)$ where:

- each node n_k in $N = \{n_j\}_1^R$ represents the k^{th} room in the floor plan with R rooms.
- each edge $e_i = (n_p, n_q)$ in $E = \{e_j\}_1^M$ denotes a connection between room p and q .

In this work, we incorporate bubble diagrams as additional conditioning in our floor plan generation process. To obtain the bubble diagram, we calculate the pair-wise proximity of rooms in each floor plan. Rooms are considered to be connected if their Manhattan distance is within a threshold (8 pixels for RPLAN and 2 pixels for ProcTHOR, accounting for dataset-specific length units). This adjacency information is stored in the "edges" field of our data representation, and we use $\mathcal{G}$ as conditioning during training to generate floor plans that respect these spatial relationships.

5.4 DStruct2Design (DS2D) LLM

In this section, we introduce our fine-tuned DS2D LLM model. In our work, we choose 8B Instruct version of LLAMA3 [21, 23, 131, 132] as our LLM of choice for its capabilities,

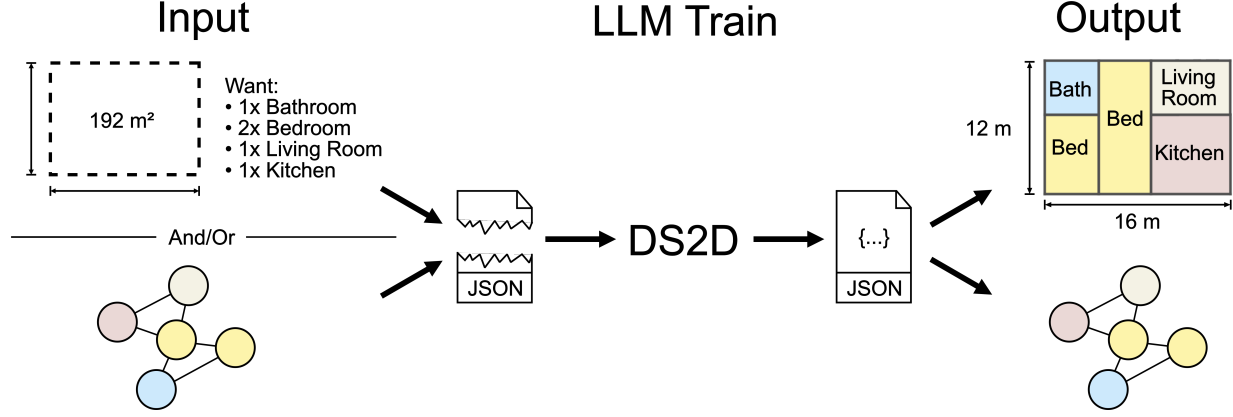


Figure 5.1 Model Pipeline Overview

smaller size and ease of use from various plug-and-play integrations that make fine-tuning on limited resources easier.

We fine-tune the model by prompting with structured numerical constraints and or graph constraints (Bubble Diagram) and asking it to predict the converted floor plan, as depicted in Figure 5.1. The key step is to construct numerical constraints and bubble diagrams into data structure format to be incorporated as part of the language prompt.

In the next sections, we explain how this process is done.

5.4.1 Numerical Constraints

Numerical constraints $\mathcal{C} = \{c_j\}_1^T$ are a set of T conditions that users may impose on the final floor plan. Any field in the floorplan data, other than explicit polygon coordinates, can be part of the constraint set. They play a crucial role in floor plan design and generation. These constraints include the total square footage of the entire floor, the number and types of rooms present, and the size of each room.

These conditions are inherently numerical, which allows us to directly leverage their data structure form in using it as part of the prompt. For example, a constraint specifying a total square footage of 2200 can be represented as `{"total_area": 2200}`. By transforming constraints into this format, we create a direct correspondence between the input prompt and the output JSON string. This approach facilitates a clearer understanding of the structure and relationships within the model, enhancing its ability to generate floor plans that adhere to the specified numerical constraints.

This direct encoding of constraints into the data structure ensures that the model can easily

interpret and apply them during the generation process. It also simplifies the validation of generated floor plans, as the output can be directly compared against the specified constraints to ensure adherence. By embedding these numerical conditions within the data structure, we streamline the fine-tuning process.

5.4.2 Bubble Diagrams

Bubble diagrams, which depict the spatial relationships and connectivity between rooms, are represented as a different form of conditioning for our model. Instead of using a visual format, we encode these diagrams as tuples of connecting rooms. For an edge $e_i = (n_p, n_q)$ that connects nodes p and q , we format it as `(room\_p, room\_q)`.

Given the potential existence of multiple rooms of the same type, which can be easily distinguished in a graph but not as readily in text, we enhance the representation of `room_p` and `room_q` by including both their room IDs and their room types. This approach ensures that each room is uniquely identifiable within the diagram.

For example, if there are two bedrooms in the floor plan, referring to them as "bedroom" would be cause confusion. By using their room IDs along with their types, such as `(bedroom 1, bedroom 2)`, we accurately describe the specific rooms that are connected. This way the structure of the bubble diagram preserved.

This method of encoding allows the model to better understand and utilize the spatial relationships and connectivity constraints during the floor plan generation process. Through this process, our model remains compatible with related work's formulation of the problem, and makes it easier for us to compare results with them, and for future work in the same vein to compare with us.

Furthermore, this approach facilitates easier integration of bubble diagrams with other numerical and categorical constraints, ensuring a comprehensive and unified input format that the model can process effectively.

5.4.3 Complete Prompt

We process the constraint set and the bubble diagram into "constraint string" and "adjacency string". They are then combined with an instruction phrase that is used throughout the training process to create the final prompt. An example of the full prompt is shown in Table 5.3

Table 5.3 Example of full instruction prompt used in training.

```

<|start_header_id|>system<|end_header_id|>
you are to generate a floor plan in a JSON structure where each room
is defined by polygon vertices, make sure to not overlap the polygons.
you have to satisfy the adjacency constraints given as pairs of
neighboring rooms; two connecting rooms, room1 and room2, are presented
as (room1_type/"room1_id", room2_type/"room2_id"). you have to also
match the specifications passed by the user in a JSON structure when
they exist. when room area and total area requirements exist, make sure
the polygon areas add up to the required number.
<|eot_id|><|start_header_id|>user<|end_header_id|>
adjacency constraints: (Bedroom/"room|4", Bathroom/"room|5"),
(Bedroom/"room|4", Kitchen/"room|6"), (Bedroom/"room|4",
LivingRoom/"room|7"), (Bathroom/"room|5", Kitchen/"room|6"),
(Bathroom/"room|5", LivingRoom/"room|7"),
(Kitchen/"room|6", LivingRoom/"room|7"). specifications:
{'room_count': 4, 'total_area': 146.8, 'rooms': [
{'area': 41.3, 'id': 'room|4', 'room_type': 'Bedroom'},
{'area': 27.5, 'id': 'room|7', 'room_type': 'LivingRoom'}]}
<|eot_id|><|start_header_id|>assistant<|end_header_id|>

```

5.5 Metrics and Benchmarks

Assessing the quality of generated floorplans is crucial that they meet the control design criteria and are functional. We focus on numerical consistency as a key aspect of floorplan quality, dividing our evaluation metrics into two categories: self-consistency and prompt-consistency.

5.5.1 Self Consistency

Self Consistency metrics measures how the numbers in a generated floorplan agree with one another within itself. An example of this is a metric to check if the area defined by the polygon vertices is the same as the area number generated in the "room_area" field.

5.5.2 Prompt Consistency

Prompt Consistency metrics evaluate how consistent the numbers in a generation are with the constraints used in the prompt. An example of this type of metrics is one that measures

Table 5.4 Metrics and Associated Acronyms Used to Evaluate Generation Quality

Metric Type	Metric	Explanation
Self Consistency (SC)	† Total Area	percentage difference between stated total area vs. the sum of all rooms' state area
	‡ Polygon Area (P. Area)	percentage difference between a room's stated area vs. area calculated from polygon
	‡ Overlap	boolean check for existence of polygon overlap
	‡ Room ID (ID)	boolean check for duplicate room id
	‡ Room Count (R. Count)	boolean check to see if "room_count" field number is equal to the number of rooms in the floor plan.
Both SC and PC	◁ Room Height (R. H)	percentage difference between height obtained from polygon vs. stated height in generation (SC) or requested height (PC).
	▷ Room Width (R. W)	percentage difference between width obtained from polygon vs. stated width in generation (SC) or requested width (PC).
Prompt Consistency (PC)	• Total Area	percentage difference between the sum of room polygon areas vs. requested total area
	♣ Num. Room (Num. R.)	percentage difference between number of rooms in the floorplan vs. requested number of rooms in the prompt.
	♠ Room ID (ID)	precision and recall of room ids in the floorplan with respect to the ids present in the prompt
	♡ Room Area (R. Area)	percentage difference between a room's polygon area vs. requested room area in the prompt
	◇ Room Type (Type)	precision and recall of room types in the floorplan with respect to the room types present in the prompt
	○ ID-Type Match (IDvsType)	percentage difference between the number of rooms with correctly matching type and id in the floor plan vs. the total number of rooms present in the prompt
Bubble Graph	Compatibility	graph edit distance (GED) between the input bubble diagram and one extracted from the output floor plan.

if the generated number of rooms adds up to the number of room requested in the prompt.

5.5.3 Summary

Together, these two groups of metrics provide a comprehensive evaluation framework for assessing the quality of generated floor plans. Self-consistency metrics ensure internal numerical

coherence, while prompt-consistency metrics verify adherence to user-specified constraints. This dual approach allows for a thorough assessment of the floor plan generation process, ensuring that the results are both accurate and relevant to the user’s needs.

In addition to the two main groups of metrics, we also incorporate **Compatibility** metrics which is used in past work [133, 118, 119, 134, 135] to measure similarity between the input bubble diagram and the output floor plan. It is by definition the graph edit distance [136] between the input bubble diagram and the output diagram extracted from the output JSON. The extraction method is the same one used to generate the bubble diagrams in the first place as described in Section 5.3.3. The full array of metrics and their explanations are presented in Table 5.4.

5.6 Experiments

All of our experiments are ran on the LLaMA3-8B-Instruct variant of the LLaMA model family [21, 23, 132]. We train our models by running 8-bit quantization along with LoRA [137] explained in Section 2.2.5. The exact training parameters and data split details are discussed in the next section.

5.6.1 Training Details

For floorplan data converted from **RPLAN** dataset, we follow the original data split. RPLAN dataset is divided into 5 parts according to the number of rooms present in the floorplan. Specifically, each part respectively contains floorplans with 4, 5, 6, 7, 8 rooms. Each part is further divided into a training, validation, and a test split. Following prior work, we use only training data with 5, 6, 7, 8 rooms. 4 different models are trained, each has never seen one of the 4 part – when training the model for 5 room floorplan generation, training is done with training splits of 6, 7, 8 room floorplans etc. During test time, the model is asked to generate only 4 room floorplans even though it has never seen a 4 room floorplan in the training set. This is in line with prior work that uses the RPLAN dataset.

For floorplan data converted from **ProcTHOR** dataset, the validation and test split is randomly chosen, with each of validation and test split being 10% of the training split. The training split is used to train the model; the validation split used to perform early stopping; and the test split is used to generate prompts for generation and evaluation.

5.6.2 Hyper-parameters

We follow the suggest training parameters from Llama-recipes (<https://github.com/meta-llama/llama-recipes/tree/main>), and we do not perform any hyper-parameter search.

Random Seed used for generation is chosen to be 12345 to select 100 (sampling generation) or 1000 (greedy generation) random test set data for generation. This seed is simply chosen and never changed or compared with any other random seed.

For **Sampling** generation, we generate 20 output floorplans for each prompt using nucleus sampling. We choose $p = 0.8$ for sampling as it is a common value for this parameter.

For **Greedy** generation, we use the default parameter provided by HuggingFace hug [138].

Table 5.5 Summary of model variants

	ProcTHOR Models						RPLAN Models			
	F	M	PM	F+BD	M+BD	PM+BD	5-R	6-R	7-R	8-R
BD	-	-	-	✓	✓	✓	✓	✓	✓	✓
Masking	-	random	preset	-	random	preset	random	random	random	random

5.6.3 Model Variants

RPLAN

Previous research [118, 119, 120] using RPLAN has employed a strategy to evaluate the generative model’s ability to generalize out-of-distribution by training separate model variants based on the total number of rooms. The dataset is divided into four subsets, each containing floorplans with 5, 6, 7, or 8 rooms. Then, a model is trained for each combination of three subsets, excluding one subset each time.

For instance, one model is trained on floorplans with 6, 7, and 8 rooms, while another is trained on floorplans with 5, 6, and 7 rooms. During inference, to generate a 5-room floorplan, the model trained without 5-room floorplans is used. This approach assesses the model’s ability to generalize to unseen room counts.

Adopting this approach, we train four distinct model variants on RPLAN, denoted as 4-R, 5-R, 6-R, and 7-R. The number in each model name indicates the number of rooms in the floorplans it generates, and also signifies that the model was trained without floorplans of that specific room count. This training strategy enables us directly with prior work in the field.

ProcTHOR

Simulating real-world user requests requires training model variants on ProcTHOR with diverse constraint sets. We investigate robustness to missing information by training three variants with different constraint sets. As a reminder, any field in the floorplan data, other than explicit polygon coordinates, can be part of the constraint set. For each variant, we train two models: a Bubble Diagram Enabled Model (BD) and a Numerical Constraint Only Model.

The BD model incorporates Bubble Diagrams as input, utilizing adjacency strings described in Section 5.4. In contrast, Numerical Constraint Only Models rely solely on constraint strings as input. This approach yields six distinct model variants trained on ProcTHOR, enabling us to evaluate their performance and adaptability in various scenarios.

By training these variants, we can assess how well the models generalize and respond to different constraint sets, mimicking real-world user requests that may vary in completeness and specificity. This comprehensive approach allows us to evaluate the models' robustness and practical applicability in handling diverse user requests.

The three main variants are defined by the constraint set used, and they are listed below:

- **Full-Prompt (F)** model are trained using full constraint set. Every attribute of floorplans, other than specific polygon coordinates, are used as conditioning. Essentially, the only degree of freedom left to the model is the polygon placement. Every other attribute, such as room size, height, width, type, and in the case of the BD variant, room connectivity, are all specified.
- **Mask (M)** model uses a subset of constraint set by applying a 50% masking on every single possible constraint. In the case of list data field, such as `rooms`, the masking is applied independently to individual item in the list. (`rooms` is only dropped out if the entire list becomes empty.) As a safety measure, we always keep at least 1 constraint in the set.
- **Preset Mask (PM)** randomly selects one of four preset constraint sets with varying degree of missing information. The idea is to not just have IID random masking, but a hierarchy of attributes from general to specific which are detailed in Table 5.6.

The summary of our six variants and their abbreviations are shown in Table 5.5. The "+BD" suffix indicate BD variants. The ProcTHOR trained model without "+BD" suffix are numerical constraint only models

5.7 Specific Attributes Sets in Preset Masking

Here we present the attributes used in the preset masking variant of our model. The four presets have varying amount of attributes from general to specific.

Table 5.6 Different presets of constraint set. Partial list represents information for only some of the rooms: dropout is applied on the individual room in the list of all rooms

Preset	Constraint Attributes
1	room_count, room_types, total_area
2	room_count, room_types, total_area, partial list of (rooms[id], rooms[type], rooms[area])
3	room_count, room_types, partial list of (rooms[id], rooms[type], rooms[area])
4	room_count, room_types, full list of (rooms[id], rooms[type], rooms[area])

5.7.1 Generation Prompts

During inference, we employ four diverse generation prompts with varying levels of constraints to simulate real-world floorplan design scenarios. This approach mirrors the common practice where designers receive incomplete criteria, allowing them to exercise creativity and flexibility. For instance, a user might specify sizes for the living room and bedroom, while leaving the designer to envision various layouts with different sizes for the remaining rooms.

- **Specific (S)** prompts contains all the possible constraints, it mirrors the prompts used to train the Full-Prompt model variants.
- **All Room Area (AR)** prompts is one where we pass in the area of all the rooms (total area can be inferred from this information).
- **Total Area (TA)** prompts contain only the total area of the floor plan as a constraint.
- **Partial Room Area (PR)** prompts are similar to Total Area prompts. In addition to the total area of the floor plan, the area of some of the rooms are also passed as conditions in the constraint string.

For all of the Bubble Diagram model variants, Bubble Diagrams are used in addition to the constraint set in each of these types of prompts.

5.8 Results

We test floor plan generation quality on ProcTHOR-trained six model variants with the four different generation prompts with self and prompt consistency metrics. Results from our best



Figure 5.2 Example generations and extracted bubble diagrams. Pink background: ground truth floor plan and its associated bubble diagram. Green background: from F+BD model with Specific prompts. Blue background: from F+BD model with Total Area prompts.

model variants are listed in Table 5.7, and all results on all 48 sets of experiments are listed in Appendix B.

We perform the same evaluation on the four RPLAN-trained model variants. Because each model variant trained on RPLAN uses a slightly different set of data (following prior work), we show results for each model variant in Table 5.7. Results for the full 24 experiments are shown in Section B.

Analysis According to our metrics, perhaps surprisingly, our LLM models demonstrate a high level of competence in generating floorplans that are largely mathematically consistent, and are also consistent to input numerical constraint. As seen in Figure 5.2, when given full specifications, the generated rooms largely resembles ground truth in sizes, width and height. This quality is accurately reflected by our Total Area, R.H and R.W metrics. On the other hand, our metric suggests that generations can produce overlapping room layouts. An example of this is found in Figure 5.2, in the second row’s green background’s right-most floorplan: the top-left bathroom is overlapping on top of the bedroom. This problem is especially apparent in the RPLAN-trained models, and is accurately reflected by the P. Overlap metrics in Table 5.7. This limitation can perhaps be tackled in future work.

Table 5.7 Our self- and prompt-consistency Benchmark results. Subscripts under model variants are generation prompts.

	Self-consistency Benchmark						
	P. Area [‡]	Total Area [‡]	P. Overlap [#]	R. Count [‡]	ID [‡]	R. H [◁]	R. W [▷]
F _S	0.94±0.07	1.00±0.03	0.16±0.37	1.00	1.00	0.99±0.02	0.99±0.02
M _{AR}	0.90±0.07	0.99±0.03	0.12±0.33	1.00	1.00	0.99±0.02	0.99±0.01
PM _{PR}	0.92±0.06	0.97±0.07	0.15±0.35	0.96	1.00	1.00±0.01	1.00±0.01
PM _{TA}	0.93±0.05	0.99±0.04	0.14±0.35	0.95	1.00	1.00±0.01	1.00±0.01
5-R _{AR}	0.90±0.16	0.98±0.05	0.37±0.48	0.97	1.00	1.00±0.01	0.98±0.02
6-R _S	0.93±0.05	0.98±0.06	0.36±0.48	0.99	1.00	0.99±0.01	0.97±0.02
7-R _S	0.93±0.04	0.98±0.05	0.42±0.49	0.97	1.00	0.99±0.01	0.98±0.02
8-R _{TA}	0.82±3.50	0.97±0.07	0.53±0.50	0.89	1.00	0.99±0.01	0.98±0.02

	Prompt-consistency Benchmark						
	Num. R [♣]	Total Area [♣]	R. Area [♡]	ID [♣]	IDvsType [◦]	R. H [◁]	R. W [▷]
F _S	1.00±0.00	0.95±0.07	0.94±0.07	1.00±0.03	1.00±0.00	0.99±0.02	0.99±0.02
M _{AR}	1.00±0.00	-	0.90±0.07	1.00±0.00	1.00±0.00	-	-
PM _{PR}	1.00±0.00	0.93±0.07	0.91±0.11	1.00±0.03	1.00±0.00	-	-
PM _{TA}	1.00±0.00	0.95±0.06	-	-	-	-	-
5-R _{AR}	-	-	0.92±0.04	1.00±0.01	1.00±0.00	-	-
6-R _S	1.00±0.00	0.94±0.07	0.94±0.06	1.00±0.05	1.00±0.00	0.92±0.23	0.94±0.14
7-R _S	1.00±0.00	0.94±0.06	0.94±0.06	1.00±0.02	1.00±0.00	0.93±0.17	0.94±0.29
8-R _{TA}	-	0.93±0.07	-	-	-	-	-

To compare with prior work that focus on using bubble diagrams as input on the RPLAN dataset, we prompt our RPLAN-trained models with bubble diagram. For evaluation we compute the standard compatibility metric obtained from graph edit distance calculation between the input bubble diagrams and the ones extracted from generations. Results are shown in Table 5.8. We also evaluate compatibility metrics on the ProcTHOR-trained BD model variants. The results are shown in Table 5.10.

Analysis Overall, our RPLAN-trained model generates results that are competitive with the state-of-the-art image based algorithms, even though our focus here has been on numerical accuracy. One caveat is that the bad generations are very wrong, causing our method to have a much higher error rate compared to image-based methods. Table 5.10 suggests our model has a harder time following the input bubble diagram as the number of room increases, or when given room area information. This is reflected in Figure 5.2 where the second row’s bubble diagrams show more variations. This limitation might be tackled by using a better data structure to represent bubble diagrams, and perhaps also to be included as part of

Table 5.8 Comparison of Compatibility↓ score (Graph Edit Distance) on the RPLAN dataset.

Model	Number of Rooms in Floorplan			
	5	6	7	8
Ashual and Wolf [135]	7.5 ± 0.0	9.2 ± 0.0	10.0 ± 0.0	11.8 ± 0.0
Johnson et al. [134]	7.7 ± 0.0	6.5 ± 0.0	10.2 ± 0.0	11.3 ± 0.1
House-GAN [128]	2.5 ± 0.1	2.4 ± 0.1	3.2 ± 0.0	5.3 ± 0.0
House-GAN++ [129]	1.9 ± 0.3	2.2 ± 0.3	2.4 ± 0.3	3.9 ± 0.5
HouseDiffusion [120]	1.5 ± 0.0	1.2 ± 0.0	1.7 ± 0.0	2.5 ± 0.0
Our BD Model w/ BD prompt	0.46 ± 0.73	0.79 ± 0.98	1.27 ± 1.28	2.50 ± 1.97

the floorplan data representation. This way, the bubble diagram will simply be part of the constraint set, and this may help the model understand the input better. This can be explored in future work in the same task setting. Additionally, in the future, our data can be expanded to account for more floorplan attributes such as doors and walls and our benchmark can include additional metrics to account for more obscure numerical checks.

Table 5.9 Our self- and prompt-consistency on few-shot results

	Self-consistency Benchmark						
	P. Area [‡]	Total Area [‡]	P. Overlap [#]	R. Count [‡]	ID ^b	R. H [◁]	R. W [▷]
Few-Shots _S	0.99±0.00	1.00±0.00	0.04±0.20	1.00	1.00	0.93±0.02	0.93±0.01
Few-Shot _{AR}	0.99±0.01	0.96±0.07	0.01±0.11	1.00	1.00	0.88±0.07	0.93±0.00
Few-Shot _{PR}	1.00±0.00	0.94±0.12	0.00±0.04	0.70	1.00	0.86±0.07	0.93±0.00
Few-Shot _{TA}	1.00±0.00	1.00±0.00	0.00±0.00	1.00	1.00	0.93±0.00	0.93±0.00
	Prompt-consistency Benchmark						
	Num. R [♣]	Total Area [•]	R. Area [♡]	ID [♠]	IDvsType [◦]	R. H [◁]	R. W [▷]
Few-Shots _S	0.82±0.39	1.00±0.00	0.99±0.00	0.82±0.39	1.00±0.00	0.93±0.02	0.93±0.01
Few-Shot _{AR}	0.95±0.22	-	0.99±0.01	0.95±0.22	1.00±0.00	-	-
Few-Shot _{PR}	0.98±0.14	1.00±0.00	1.00±0.00	0.98±0.14	1.00±0.00	-	-
Few-Shot _{TA}	1.00±0.03	1.00±0.00	-	-	-	-	-

Analysis We further present few-shot results with GPT4o-mini. This is done by prompting the LLM with multiple example specifications along with output floorplan Jsons. The generated floorplan has less overlapping polygons compared to the ones generated by the fine-tuned LLAMA3 models, however, they are less accurate in following the size, height and width specifications.

Table 5.10 Compatibility↓ score on ProcTHOR-trained models.

Model	Generation	Number of Rooms				
Variant	Prompt	3	4	5	6	7
F+BD	Specific	0.15 $\pm$ 0.41	0.33 $\pm$ 0.58	1.35 $\pm$ 1.34	2.24 $\pm$ 1.76	4.23 $\pm$ 1.89
	All Room Area	5.91 $\pm$ 0.29	9.75 $\pm$ 0.54	13.79 $\pm$ 1.15	18.09 $\pm$ 1.71	22.35 $\pm$ 2.24
	Partial Room Area	3.80 $\pm$ 1.20	5.73 $\pm$ 2.03	8.19 $\pm$ 2.72	10.15 $\pm$ 2.88	11.42 $\pm$ 4.52
	Total Area	0.19 $\pm$ 0.40	0.62 $\pm$ 0.78	3.48 $\pm$ 1.83	3.41 $\pm$ 1.50	6.23 $\pm$ 2.20
M+BD	Specific	0.16 $\pm$ 0.45	0.47 $\pm$ 0.67	2.06 $\pm$ 1.33	2.85 $\pm$ 1.79	5.23 $\pm$ 2.14
	All Room Area	5.91 $\pm$ 0.29	9.62 $\pm$ 0.56	12.90 $\pm$ 1.27	17.48 $\pm$ 1.82	22.13 $\pm$ 1.94
	Partial Room Area	3.85 $\pm$ 1.11	6.35 $\pm$ 1.95	8.44 $\pm$ 2.48	11.56 $\pm$ 3.42	14.13 $\pm$ 3.90
	Total Area	0.20 $\pm$ 0.40	0.54 $\pm$ 0.73	2.40 $\pm$ 1.03	3.09 $\pm$ 2.07	6.26 $\pm$ 2.03
PM+BD	Specific	0.15 $\pm$ 0.39	0.58 $\pm$ 0.96	2.42 $\pm$ 1.33	2.97 $\pm$ 1.62	5.97 $\pm$ 1.25
	All Room Area	5.88 $\pm$ 0.33	9.77 $\pm$ 0.49	13.17 $\pm$ 1.23	18.56 $\pm$ 1.58	21.97 $\pm$ 2.36
	Partial Room Area	3.87 $\pm$ 1.13	6.42 $\pm$ 1.97	8.99 $\pm$ 2.67	12.06 $\pm$ 3.05	13.68 $\pm$ 3.71
	Total Area	0.34 $\pm$ 0.93	0.46 $\pm$ 0.84	2.08 $\pm$ 1.29	2.65 $\pm$ 1.87	5.84 $\pm$ 1.61

5.8.1 Creativity In Designs

As shown in Figure 5.2, the floorplans generated by our model exhibit certain degree of creativity. While it is hard to say whether it can be classified as a particular style of expression – as sought out by our main pursuit of true artificial intelligence, it is clear that the model is able to generate an array of designs that all follow the given constraints, much like a human designer.

5.9 Conclusion

We have motivated the need for new datasets and benchmarks for real-world use case scenarios for floorplan generation. We have developed and explored a Llama 3 based LLM for the problem and it yields SOTA results on the previous compatibility based evaluation and along with our proposed metrics, it highlights different use cases scenarios where improvements are possible. In particular we see from Table 5.10 that when conditioning on bubble diagrams and room area information this model struggles to respect the bubble diagram constraints. We also see that the biggest weakness of this otherwise SOTA model is linked to issues with generating overlapping rooms.

We hope that our data, simulated data generation procedure and benchmarks will stimulate further developments for this new problem formulation of floorplan generation.

CHAPTER 6 CONCLUSION

In this thesis, we explore generative models and controlled generation. We aim to improve generation quality in both text and video, and to better align the generations with the control input.

First, we push the limit of light weight CNN-based vision models by improving its ability to remember past conditions and removing a big source of compounding errors in the generation pipeline. Second, we move on to popular diffusion based models and introduce a novel way to incorporate bounding box sequences as a mode of control in the diffusion video generation process. In doing so, we develop a two component approach that takes care of both predicting plausible bounding box sequences and generating realistic videos that strictly follows the given bounding box sequences. We show that in our approach, video generation is indeed controllable and it adheres well to the given control input. Third, we reformulate the floorplan design problem into a data structure based text generation problem. In our formulation we design a image to text data structure conversion process that works on two major source of floorplan data, and we present a suite of metrics to evaluate the generated floorplan both on its numerical consistency and on its adherence to the control input. We show that our baseline LLM model is able to create valid floorplans that comply with the given control constraints.

In the next sections, we discuss some of the challenges we recognize in our works, and possible future research directions that are natural continuations to what we have already done.

6.1 Cumulative Error from Iterative Process

Cumulative error has always existed in iterative processes. It is the same cause that leads to the famous exploding and vanishing gradient problem in recurrent networks like RNNs. In the same vein, in video generation models that employs an iterative approach to generating the frames, it causes the video to become increasing blurry.

This issue is especially apparent when we review SimVP and also during our research on ViP-STEAM. SimVP takes a block to block approach in generating video frames. This means it starts with a set amount of frames, say N frames, or a "block", and generate the next block of N frames conditioned on the current block. Then it uses the generated block as condition to generate the block after. As one can imagine, and blurriness or inaccuracies from the first generated block simply gets carried over to the next, and the inaccuracies can only build up

which results in a quick degradation of generation quality.

In ViP-STEAM(Chapter 3), even though we introduce a shift of this block-to-block process from the image space into the latent space – which effectively by-pass the process encoding, decoding, and re-encoding of frames to reduce this cumulative error – it does not completely rid the generation of this problem. In fact, in the examples we present in Section 3.4.4, this phenomenon can be observed. The person walking in and out of the view become slightly blurry in the middle of the video and as the frame progresses the blur becomes increasingly noticeable.

On the other hand, video diffusion models treat video generation as a one-step generation problem. It process whole videos as a big image by rolling out the time dimension of the video. It also generates video with the same approach. Since every frame of the video is generated concurrently with the other frames, there is no iterative process like the one in our CNN-based model, and the video quality is much higher and more consistent.

However as video clip grows in length in the future, to tens of minutes, or even hours in length, an iterative process may become necessary, and this cumulative error problem must be addressed to maintain the same high quality of short video generations.

6.2 Limit Number of Bounding Boxes

During our review for current bounding box controlled video generation methods, we are surprised to find that all of the existing methods deal with only a very limited amount of bounding boxes. Most method either focus on single bounding box object or has a method to filter out bound boxes with strict criteria such as persistence throughout the entire video, which leads to very few candidates. In our work, we do not impose selective criteria on which bounding boxes to use. However in order to avoid erroneous bounding boxes either in labeling or position, especially in the vKitti dataset where there are a lot of bounding boxes with no object in them, we set a number threshold to the amount of total bounding boxes used in training. In our case, that threshold is set to 15.

This limitation does not necessarily mean the model cannot generate consistent videos when controlled by frames with more than 15 bounding boxes. It does mean that the BBox Predictor is inclined to generate bounding box frames with only 15 or less bounding boxes. Overall, this can be viewed as masking on data with more than 15 bounding boxes. Nevertheless it is an limitation of our model, but it can be easily improved on in the future.

6.3 Video Generation Length

Alluded to in the previous sections, current video generation methods tend to only work well when creating short clips. In demos of current approaches, clips usually last only seconds, some even start being inconsistent within a second. A big limiting factor is the resources required for video projects. Processing and generating long videos take a lot more resources than we have and, in general, scaling up this process might require improved model architectures for support. As previously mentioned, this in turn could be new problems associated with generating lengthy videos.

6.4 Latent Space Exploration

One area that is unexplored is driving video and text generations using latent space. As we have seen in ViP-STEAM (Chapter 3), it remains a challenge for our model to capture stochasticity even with the help of a VAE. However VAEs are notoriously difficult to train and perhaps a more careful approach to training the VAE may lead to different results. Regardless, future research could focus on using an latent space method such as VAE to better capture stochasticity in video data, in order to generate higher quality videos without mode collapsing.

6.5 Text Visualization in Videos

It is a well known issue that video generation models often cannot correctly visualize texts in scenes. There are specialized models that tackle this specific issue, however in most video generation models, including our Ctrl-V model, this is still a notable limitation. This becomes very noticeable when the generated scene involves any signage, road side shop front or painted texts in the road lanes. In most cases, the text would take shape similar to letters but are not valid or recognizable. They also often morph into less comprehensible shapes throughout the course of the clip. To demonstrate, we include a couple examples in Figure 6.1 and Figure 6.2. The "One Way" text on the sign is intelligible in the generated frame in Figure 6.1; The ground truth is provided for comparison. In Figure 6.2, the "ONLY" text in the generated video progressive gets worse visually. It starts with half of the word being readable, then becomes more dim, and at the end it morphs into something incomprehensible.

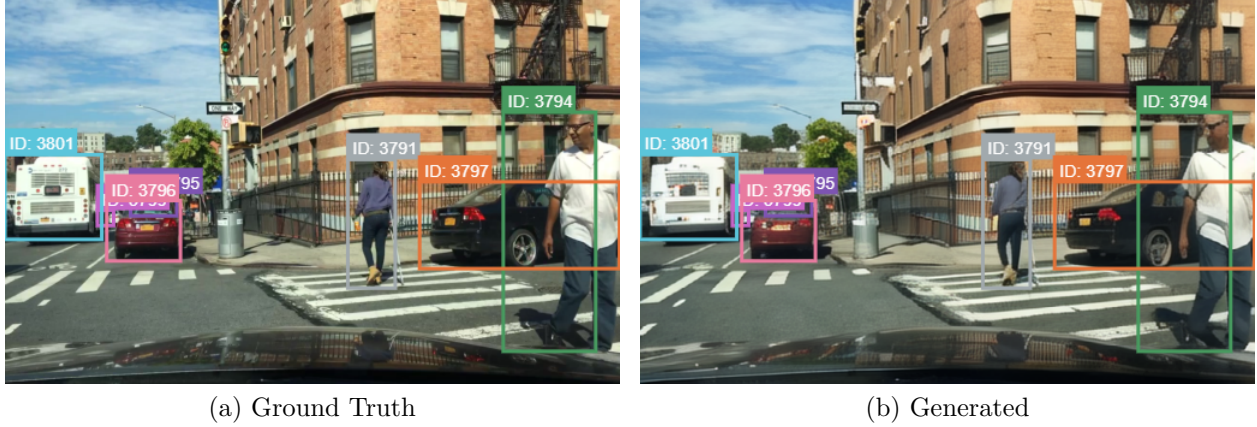


Figure 6.1 Comparison of One-Way sign between ground truth and generated video frame.

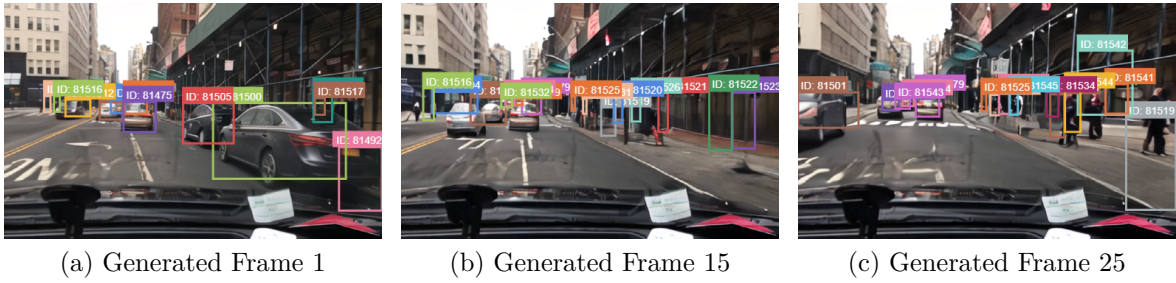


Figure 6.2 Text in the driving lane morphs over time.

6.6 Invalid Generations

In our DS2D work (Chapter 5), we transform floorplan into a strict JSON-like data structure, and flatten it in plain text. Because no explicit regularizer is used to enforce the text structure, there are cases where the generated floorplans are invalid. The invalid generations mostly fail into one of two categories: structural integrity or content integrity.

In the case of failing structural integrity, the generated floorplan may contain empty fields or, in rare cases, run on and on with repeated field entries. This closely relates to the classic problem of language model "hallucination" where the model can output invalid or unrelated, or even falsely-informed text during generation. Nevertheless, this problem can be mitigated with more training data and perhaps different fine-tuning strategy.

The case of generated floorplans failing content integrity is a harder one to tackle. Even though the floorplans generated by our LLM shows good numerical consistency both within the floorplan and with respect to the control numerical constraints, there are cases where the polygon placement overlap with each other. We show one such case in Figure 6.3.

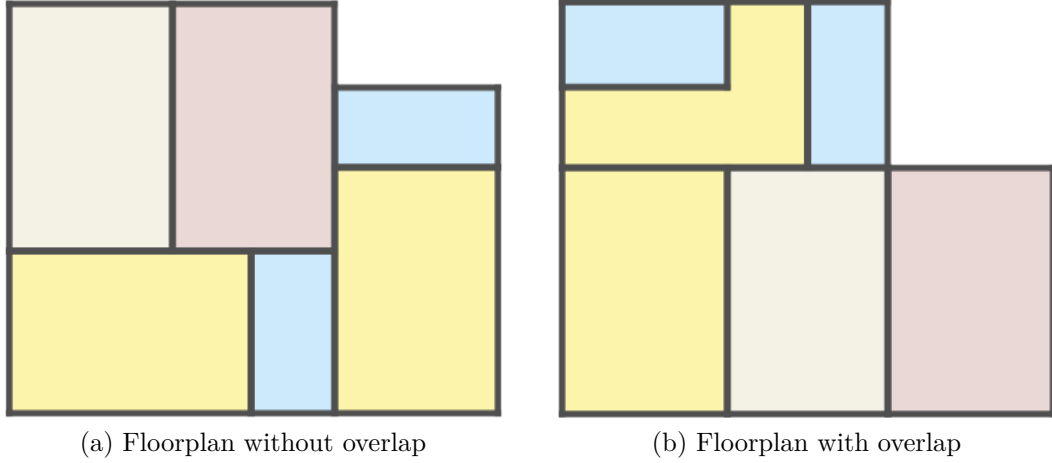


Figure 6.3 Floorplan generated with the same specification.

The two floorplans are generations sampled from the same input prompt. All of the rooms' areas are specified as constraint. Both floorplans in fact follow the constraint very well and each room in one floorplan in fact has the same area as its counterpart in the other. However we can see, in the floorplan on the right, the top bedroom (in yellow) shows an L-shaped layout with much less area than the bedrooms in the left floorplan. But by examining the generated text from which this image is converted, the two bedrooms on the right indeed should have the same area as those on the left. This discrepancy is the result of the top left bathroom (in blue, in Figure 6.3(b)) being placed right on top of the bedroom. Originally, the bedroom takes a rectangular shape that happens to include the entirety of the bathroom; and that would make its area match one of the bedrooms in the other floorplan.

When we use sampling generation strategy such as nucleus sampling, this limitation is not too big of an issue as most samples are valid. But there is no guarantee when we use greedy strategy to generate the floorplans. This is a problem that needs to be tackled in the future.

6.7 Future Research

Overall our work presented in this thesis has several direct follow up research directions.

6.7.1 ViP-STEAM Follow Up

In terms of light weight CNN-based video generation models, it is important to continue to improve the generation quality of the generations by reducing accrued inaccuracies. This could be tackled in several possible directions: introducing new mechanics to always refer

generations to the initial frames which have no blurring or accumulated errors; implement methods of segmentation to identify and extract objects as means to anchor the generations.

6.7.2 Ctrl-V Follow Up

As follow up to Ctrl-V, there are multiple directions we would like to pursue. We would like to incorporate even simpler control modes using the existing architecture. for instance, in the work presented here we showed our initial experiments with using object midpoint traces in place of bounding boxes and showed competitive metrics on the generations using those traces. This can be easily expanded to replace the entire bounding box sequence with traces, which allows more flexibility while making it cheaper to edit the control inputs. It may even be a base to build models that allow on-the-fly edits to the traces. On the other hand, we also want to build on top of Ctrl-V to introduce a unified method of taking different control inputs (optical flow, depth map, etc). Current work on controlled video generation using Stable Video Diffusion mostly introduce novel adapters specific to the input. However, Ctrl-V shows it is effective to render control input to image to allow the use of general ControlNet adapter. This approach can be explored to incorporate other modes of control input to allow flexible conditioning in video generation.

6.7.3 DS2D Follow Up

As previously mentioned, floorplan generation can some time produces overlapping room layouts. This challenge can be tackled in future work to increase the percentage of valid floorplans generated.

On a higher level perspective, we can explore using this data-structure language approach on more design problems.

REFERENCES

- [1] Jiawei Wang, Yuchen Zhang, Jiaxin Zou, Yan Zeng, Guoqiang Wei, Liping Yuan, and Hang Li. Boximator: Generating rich and controllable motions for video synthesis, 2024.
- [2] Pengxiang Li, Kai Chen, Zhili Liu, Ruiyuan Gao, Lanqing Hong, Guo Zhou, Hua Yao, Dit-Yan Yeung, Huchuan Lu, and Xu Jia. Trackdiffusion: Tracklet-conditioned video generation via diffusion models, 2024.
- [3] Cheng Tan, Zhangyang Gao, Siyuan Li, and Stan Z. Li. Simvp: Towards simple yet powerful spatiotemporal predictive learning, 2023.
- [4] Zhangyang Gao, Cheng Tan, Lirong Wu, and Stan Z. Li. Simvp: Simpler yet better video prediction, 2022.
- [5] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. ISSN 00264423. URL <http://www.jstor.org/stable/2251299>.
- [6] Marvin L. Minsky, Nathaniel Rochester, Claude E. Shannon, and John McCarthy. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. 27(4):12, 2006. ISSN 2371-9621. doi: 10.1609/aimag.v27i4.1904. URL <https://aaai.org/ojs/index.php/aimagazine/article/view/1904>.
- [7] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL <http://arxiv.org/abs/1712.01815>.
- [8] Joseph Weizenbaum. Eliza a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966. ISSN 0001-0782. doi: 10.1145/365153.365168. URL <http://doi.acm.org/10.1145/365153.365168>.
- [9] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018. URL <https://api.semanticscholar.org/CorpusID:49313245>.

- [10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.
- [11] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. URL <https://arxiv.org/abs/1312.6114>.
- [12] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. doi: 10.1073/pnas.79.8.2554. URL <https://www.pnas.org/doi/abs/10.1073/pnas.79.8.2554>.
- [13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [15] Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and Philip S Yu. Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/e5f6ad6ce374177eef023bf5d0c018b6-Paper.pdf.
- [16] Xingjian SHI, Zhouong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wangchun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf.
- [17] Yunbo Wang, Lu Jiang, Ming hsuan Yang, Jia Li, Mingsheng Long, and Fei-Fei Li. Eidetic 3d lstm: A model for video prediction and beyond. In *ICLR*, 2019.

- [18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL <http://arxiv.org/abs/1706.03762>.
- [19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019. URL <http://arxiv.org/abs/1910.10683>.
- [20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL <http://arxiv.org/abs/1810.04805>.
- [21] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.
- [22] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021. URL <https://arxiv.org/abs/2104.09864>.
- [23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

- [24] Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *CoRR*, abs/2108.12409, 2021. URL <https://arxiv.org/abs/2108.12409>.
- [25] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. CTRL: A conditional transformer language model for controllable generation. *CoRR*, abs/1909.05858, 2019. URL <http://arxiv.org/abs/1909.05858>.
- [26] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *CoRR*, abs/2101.00190, 2021. URL <https://arxiv.org/abs/2101.00190>.
- [27] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. URL <https://arxiv.org/abs/2106.09685>.
- [28] Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram A. Saletore. Efficient 8-bit quantization of transformer neural machine language translation model. *CoRR*, abs/1906.00532, 2019. URL <http://arxiv.org/abs/1906.00532>.
- [29] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2024. URL <https://arxiv.org/abs/2306.00978>.
- [30] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL <https://arxiv.org/abs/2210.17323>.
- [31] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.
- [32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

- [33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL <http://arxiv.org/abs/1505.04597>.
- [34] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018. URL <http://arxiv.org/abs/1803.08375>.
- [35] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL <http://arxiv.org/abs/1409.4842>.
- [36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [37] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020. URL <https://arxiv.org/abs/2006.11239>.
- [38] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models, 2023.
- [39] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015. URL <http://arxiv.org/abs/1503.03585>.
- [40] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. *CoRR*, abs/2102.09672, 2021. URL <https://arxiv.org/abs/2102.09672>.
- [41] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, Varun Jampani, and Robin Rombach. Stable video diffusion: Scaling latent video diffusion models to large datasets, 2023.
- [42] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, Varun Jampani, and Robin Rombach. Stable video diffusion: Scaling latent video diffusion models to large datasets, 2023. URL <https://arxiv.org/abs/2311.15127>.
- [43] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023.

- [44] Hsu-Kuang Chiu, Ehsan Adeli, and Juan Carlos Niebles. Segmenting the future. *CoRR*, abs/1904.10666, 2019. URL <http://arxiv.org/abs/1904.10666>.
- [45] Osamu Shouno. Photo-realistic video prediction on natural videos of largely changing frames. *CoRR*, abs/2003.08635, 2020. URL <https://arxiv.org/abs/2003.08635>.
- [46] Ziru Xu, Yunbo Wang, Mingsheng Long, and Jianmin Wang. Predcnn: Predictive learning with cascade convolutions. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2940–2947. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/408. URL <https://doi.org/10.24963/ijcai.2018/408>.
- [47] Ziwei Liu, Raymond A. Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow, 2017.
- [48] Hang Gao, Huazhe Xu, Qi-Zhi Cai, Ruth Wang, Fisher Yu, and Trevor Darrell. Disentangling propagation and generation for video prediction, 2019.
- [49] Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. *CoRR*, abs/1802.07687, 2018. URL <http://arxiv.org/abs/1802.07687>.
- [50] Lluís Castrejón, Nicolas Ballas, and Aaron C. Courville. Improved conditional vrnnns for video prediction. *CoRR*, abs/1904.12165, 2019. URL <http://arxiv.org/abs/1904.12165>.
- [51] Vincent Le Guen and Nicolas Thome. Disentangling physical dynamics from unknown factors for unsupervised video prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [52] Wei Yu, Yichao Lu, Steve Easterbrook, and Sanja Fidler. Efficient and information-preserving future frame prediction and beyond. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1eY_pVYvB.
- [53] Sangmin Lee, Hak Gu Kim, Dae Hwi Choi, Hyung-Il Kim, and Yong Man Ro. Video prediction recalling long-term motion context via memory alignment learning. *CoRR*, abs/2104.00924, 2021. URL <https://arxiv.org/abs/2104.00924>.
- [54] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.
- [55] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

- [56] Vikram Voleti, Alexia Jolicoeur-Martineau, and Christopher Pal. Mcvd: Masked conditional video diffusion for prediction, generation, and interpolation, 2022.
- [57] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf.
- [58] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W. Cottrell, and Julian J. McAuley. Rezero is all you need: Fast convergence at large depth. *CoRR*, abs/2003.04887, 2020. URL <https://arxiv.org/abs/2003.04887>.
- [59] Meng-Hao Guo, Zheng-Ning Liu, Tai-Jiang Mu, and Shi-Min Hu. Beyond self-attention: External attention using two linear layers for visual tasks. *CoRR*, abs/2105.02358, 2021. URL <https://arxiv.org/abs/2105.02358>.
- [60] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *CoRR*, abs/1405.3866, 2014. URL <http://arxiv.org/abs/1405.3866>.
- [61] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL <http://arxiv.org/abs/1704.04861>.
- [62] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *CoRR*, abs/2201.03545, 2022. URL <https://arxiv.org/abs/2201.03545>.
- [63] Meng-Hao Guo, Cheng-Ze Lu, Zheng-Ning Liu, Ming-Ming Cheng, and Shi-Min Hu. Visual attention network, 2022.
- [64] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters - improve semantic segmentation by global convolutional network. *CoRR*, abs/1703.02719, 2017. URL <http://arxiv.org/abs/1703.02719>.

- [65] Xiaohan Ding, Xiangyu Zhang, Yizhuang Zhou, Jungong Han, Guiguang Ding, and Jian Sun. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns, 2022.
- [66] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. January 2016. 4th International Conference on Learning Representations, ICLR 2016 ; Conference date: 02-05-2016 Through 04-05-2016.
- [67] Jungbeom Lee, Jangho Lee, Sungmin Lee, and Sungroh Yoon. Msnet: Mutual suppression network for disentangled video representations. *CoRR*, abs/1804.04810, 2018. URL <http://arxiv.org/abs/1804.04810>.
- [68] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.
- [69] Soo-Min Kang and Richard P. Wildes. Review of action recognition and detection methods. *CoRR*, abs/1610.06906, 2016. URL <http://arxiv.org/abs/1610.06906>.
- [70] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016. URL <http://arxiv.org/abs/1604.01685>.
- [71] Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *CoRR*, abs/1710.05268, 2017. URL <http://arxiv.org/abs/1710.05268>.
- [72] Bohan Wu, Suraj Nair, Roberto Martín-Martín, Li Fei-Fei, and Chelsea Finn. Greedy hierarchical variational autoencoders for large-scale video prediction. *CoRR*, abs/2103.04174, 2021. URL <https://arxiv.org/abs/2103.04174>.
- [73] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.
- [74] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

- [75] Cheng Tan, Siyuan Li, Zhangyang Gao, Wenfei Guan, Zedong Wang, Zicheng Liu, Lirong Wu, and Stan Z Li. Openstl: A comprehensive benchmark of spatio-temporal predictive learning. In *Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.
- [76] Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, John Alejandro Castro-Vargas, Sergio Orts-Escolano, José García Rodríguez, and Antonis A. Argyros. A review on deep learning techniques for video prediction. *CoRR*, abs/2004.05214, 2020. URL <https://arxiv.org/abs/2004.05214>.
- [77] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.
- [78] Pu Cao, Feng Zhou, Qing Song, and Lu Yang. Controllable generation with text-to-image diffusion models: A survey. *arXiv preprint arXiv:2403.04279*, 2024.
- [79] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator, 2017.
- [80] Ze Yang, Yun Chen, Jingkan Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator, 2023.
- [81] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models, 2022.
- [82] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021. URL <https://arxiv.org/abs/2103.00020>.
- [83] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [84] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.
- [85] Yingqing He, Tianyu Yang, Yong Zhang, Ying Shan, and Qifeng Chen. Latent video diffusion models for high-fidelity long video generation, 2023.

- [86] Yan Zeng, Guoqiang Wei, Jiani Zheng, Jiaxin Zou, Yang Wei, Yuchen Zhang, and Hang Li. Make pixels dance: High-dynamic video generation, 2023.
- [87] Jay Zhangjie Wu, Yixiao Ge, Xintao Wang, Weixian Lei, Yuchao Gu, Yufei Shi, Wynne Hsu, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. Tune-a-video: One-shot tuning of image diffusion models for text-to-video generation, 2023.
- [88] Xiang Wang, Hangjie Yuan, Shiwei Zhang, Dayou Chen, Jiuniu Wang, Yingya Zhang, Yujun Shen, Deli Zhao, and Jingren Zhou. Videocomposer: Compositional video synthesis with motion controllability, 2023.
- [89] Eyal Molad, Eliahu Horwitz, Dani Valevski, Alex Rav Acha, Yossi Matias, Yael Pritch, Yaniv Leviathan, and Yedid Hoshen. Dreamix: Video diffusion models are general video editors, 2023.
- [90] Duygu Ceylan, Chun-Hao Paul Huang, and Niloy J. Mitra. Pix2video: Video editing using image diffusion, 2023.
- [91] Johanna Karras, Aleksander Holynski, Ting-Chun Wang, and Ira Kemelmacher-Shlizerman. Dreampose: Fashion image-to-video synthesis via stable diffusion, 2023.
- [92] Weifeng Chen, Jie Wu, Pan Xie, Hefeng Wu, Jiashi Li, Xin Xia, Xuefeng Xiao, and Liang Lin. Control-a-video: Controllable text-to-video generation with diffusion models, 2023.
- [93] Zhihao Hu and Dong Xu. Videocontrolnet: A motion-guided video-to-video translation framework by using diffusion model with controlnet, 2023.
- [94] Ernie Chu, Shuo-Yen Lin, and Jun-Cheng Chen. Video controlnet: Towards temporally consistent synthetic-to-real video translation using conditional image diffusion models, 2023.
- [95] Yabo Zhang, Yuxiang Wei, Dongsheng Jiang, Xiaopeng Zhang, Wangmeng Zuo, and Qi Tian. Controlvideo: Training-free controllable text-to-video generation, 2023.
- [96] Chong Mou, Mingdeng Cao, Xintao Wang, Zhaoyang Zhang, Ying Shan, and Jian Zhang. Revideo: Remake a video with motion and content control, 2024.
- [97] Tobias Fischer, Thomas E. Huang, Jiangmiao Pang, Linlu Qiu, Haofeng Chen, Trevor Darrell, and Fisher Yu. Qdtrack: Quasi-dense similarity learning for appearance-only multiple object tracking, 2023.

- [98] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.
- [99] Xiaofeng Wang, Zheng Zhu, Guan Huang, Xinze Chen, Jiagang Zhu, and Jiwen Lu. Drivedreamer: Towards real-world-driven world models for autonomous driving, 2023.
- [100] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.
- [101] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual kitti 2, 2020.
- [102] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning, 2020.
- [103] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, Dhruv Nair, Sayak Paul, William Berman, Yiyi Xu, Steven Liu, and Thomas Wolf. Diffusers: State-of-the-art diffusion models. <https://github.com/huggingface/diffusers>, 2022.
- [104] Zhou Wang and Alan C. Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE Signal Processing Magazine*, 26(1):98–117, 2009. doi: 10.1109/MSP.2008.930649.
- [105] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.
- [106] Zoran Kotevski and Pece Mitrevski. Experimental comparison of psnr and ssim metrics for video quality estimation, 01 2010.
- [107] Mahmood Sharif, Lujo Bauer, and Michael K. Reiter. On the suitability of l_p -norms for creating and preventing adversarial examples. *CoRR*, abs/1802.09653, 2018. URL <http://arxiv.org/abs/1802.09653>.
- [108] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.
- [109] Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges, 2019.

- [110] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.
- [111] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.
- [112] Dillon Reis, Jordan Kupec, Jacqueline Hong, and Ahmad Daoudi. Real-time flying object detection with yolov8, 2024.
- [113] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. Msr-vtt: A large video description dataset for bridging video and language. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5288–5296, 2016. doi: 10.1109/CVPR.2016.571.
- [114] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Nieves. Activitynet: A large-scale video benchmark for human activity understanding. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–970, 2015. doi: 10.1109/CVPR.2015.7298698.
- [115] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012. URL <http://arxiv.org/abs/1212.0402>.
- [116] Linjie Yang, Yuchen Fan, and Ning Xu. Video instance segmentation. *CoRR*, abs/1905.04804, 2019. URL <https://arxiv.org/abs/1905.04804>.
- [117] Ziniu Luo and Weixin Huang. Floorplan: Vector residential floorplan adversarial generation. *Automation in Construction*, 142:104470, 2022. ISSN 0926-5805. doi: <https://doi.org/10.1016/j.autcon.2022.104470>. URL <https://www.sciencedirect.com/science/article/pii/S0926580522003430>.
- [118] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation, 2020.
- [119] Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. House-gan++: Generative adversarial layout refinement networks, 2021.
- [120] Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous

- denoising. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5466–5475, 2023.
- [121] Zehao Wen, Zichen Liu, Srinath Sridhar, and Rao Fu. Anyhome: Open-vocabulary generation of structured and textured 3d homes. *arXiv preprint arXiv:2312.06644*, 2023.
- [122] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, et al. Holodeck: Language guided generation of 3d embodied ai environments. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, volume 30, pages 20–25. IEEE/CVF, 2024.
- [123] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhang Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 38(6), 2019.
- [124] National Institute of Informatics. Lifull homes dataset, 2015.
- [125] Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. Raster-to-vector: Revisiting floorplan transformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2195–2203, 2017.
- [126] Theodoros Galanos, Antonios Liapis, and Georgios N Yannakakis. Architext: Language-driven generative architecture design. *arXiv preprint arXiv:2303.07519*, 2023.
- [127] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. In *NeurIPS*, 2022. Outstanding Paper Award.
- [128] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 162–177. Springer, 2020.
- [129] Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. House-gan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *Proceed-*

- ings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13632–13641, 2021.
- [130] Ricardo Lopes, Tim Tutenel, Ruben M Smelik, Klaas Jan De Kraker, and Rafael Bidarra. A constrained growth method for procedural floor plan generation. In *Game-ON*, 2010.
 - [131] AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.
 - [132] Huggingface llama3. <https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct>, .
 - [133] Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising, 2022.
 - [134] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs, 2018.
 - [135] Oron Ashual and Lior Wolf. Specifying object attributes and relations in interactive scene generation, 2019.
 - [136] Zeina Abu-Aisheh, Romain Raveaux, Jean-Yves Ramel, and Patrick Martineau. An exact graph edit distance algorithm for solving pattern recognition problems. In *4th International Conference on Pattern Recognition Applications and Methods 2015*, 2015.
 - [137] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.
 - [138] Huggingface generation. https://huggingface.co/docs/transformers/v4.40.2/en/main_classes/text_generation#transformers.GenerationMixin.generate, .
 - [139] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO, January 2023. URL <https://github.com/ultralytics/ultralytics>.
 - [140] Nir Aharon, Roy Orfaig, and Ben-Zion Bobrovsky. Bot-sort: Robust associations multi-pedestrian tracking. *arXiv preprint arXiv:2206.14651*, 2022.

APPENDIX A CTRL-V: ADDITIONAL RESULTS

Notations

SVD Scaling Function Definitions

As presented in Equation 4.2, λ_{skip} , λ_{out} , λ_{in} and λ_{noise} are scaling functions where λ_{in} and λ_{out} scale the input and output magnitudes of the neural network $\mathbb{U}_\theta$ being trained, λ_{skip} modulates the skip connection, and λ_{noise} maps noise level σ_t into a conditioning input for $\mathbb{U}_\theta$. More detailed information on these functions can be found in the work of Karras et al. [81].

Unless otherwise specified, we use the SVD definitions as presented in [41]. The mathematical formulations for the scaling functions are thus:

$$\begin{aligned} \lambda_{\text{out}}(\sigma) &= \frac{-\sigma}{\sqrt{\sigma^2 + 1}} & \lambda_{\text{in}}(\sigma) &= \frac{1}{\sqrt{\sigma^2 + 1}} \\ \lambda_{\text{skip}}(\sigma) &= (\sigma^2 + 1)^{-1} & \lambda_{\text{noise}}(\sigma) &= \frac{\log \sigma}{4} \end{aligned} \tag{A.1}$$

Ctrl-V Notations

In this section, we provide a table of definition for all of the Symbols notations used in our work.

Table A.1 Glossary of terms.

Symbol	Appearance	Definition
$\mathcal{E}$	Section 4.2.1, Figure 4.1	the off-the-shelf VAE Image Encoder used by SVD
$\mathcal{D}$	Section 4.2.1, Figure 4.1	the off-the-shelf VAE Image Decoder used by SVD
$\mathcal{C}_\xi$	Figure 4.1	modified ControlNet module in Box2Video
$\mathbb{D}_\theta$	Equation 4.2	denoiser network of the SVD backbone
$\mathbb{U}_\theta$	Equation 4.1, Equation 4.2	the main UNet component of $\mathbb{D}$
$\mathbf{f}$	Section 4.2.1	a sequence of frames
$\mathbf{f}^{(i)}$	Section 4.2.1	the i^{th} frame in $\mathbf{f}$
$\mathbf{z}$	Section 4.2.1	$\mathbf{z} = \mathcal{E}(\mathbf{f})$. a sequence of frames encoded by the VAE encoder
$\mathbf{z}^{(i)}$	Section 4.2.1	$\mathbf{z}^{(i)} = \mathcal{E}(\mathbf{f}^{(i)})$. the encoded i^{th} frame.
$\mathbf{z}^{(0)}$	Section 4.2.1, Equation 4.1	encoded first frame $\mathbf{f}^{(0)}$
$\mathbf{z}_{\text{pad}}^{(0)}$	Equation 4.1, Figure 4.1	padded $\mathbf{z}^{(0)}$ by repeating itself along the first dimension N times.
t	Section 4.2.1, Equation 4.1	level used by the EDM noise scheduler to determine the noise level
$\hat{\mathbf{z}}_t$	Section 4.2.1, Equation 4.1	latent representation of frames corrupted by noise level t
$\mathbf{c}^{(0)}$	Section 4.2.1, Equation 4.1	CLIP encoding of the first frame $\mathbf{f}^{(0)}$
$\mathbf{f}_{\text{bbox}}$	Figure 4.1	bounding box frames. Each frame contains pixel renderings of bounding boxes on a blank background. Used as conditioning for Box2Video
$\mathbf{b}$	Figure 4.1	latent representation of $\mathbf{f}_{\text{bbox}}$ (encoded by $\mathcal{E}$)
$\mathbf{b}^{(0)}$	Figure 4.1	latent representation of the first bounding box frame
$\mathbf{b}^{(N-1)}$	Figure 4.1	latent representation of the last bounding box frame
$\hat{\mathbf{b}}_t$	Figure 4.1	$\mathbf{b}$ corrupted by noise level t

Motion Control Assessment: YOLOv8 Object Detector Configurations

We utilize the “yolov8x” variant of the YOLOv8 model, implemented by Ultralytics [139], for object detection. We set the detector’s Non-Maximum Suppression Intersection Over Union (NMS-IoU) threshold to be 0.35 across all experiments.

Since YOLOv8 was trained to detect objects from the MS COCO dataset [110], we adjust the class labels of our dataset to match the MS COCO labels using the following mappings:

The detection resolution is maintained at the same level as the generation/training resolution – 312×520 .

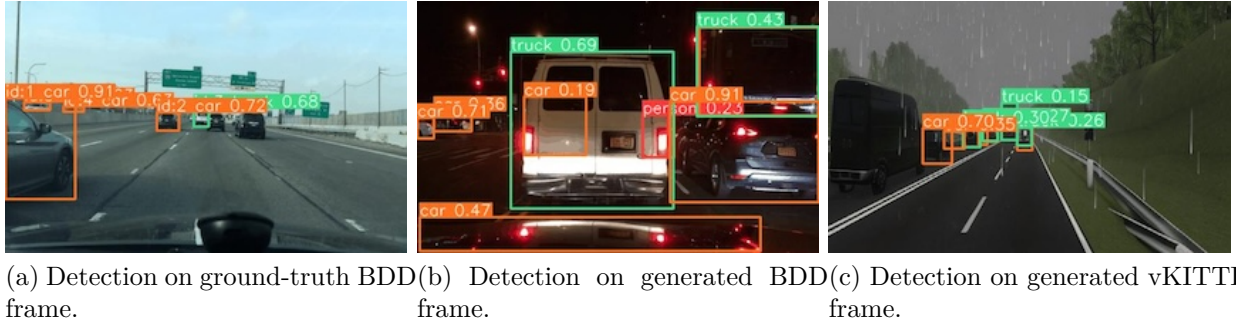


Figure A.1 YOLOv8 detections on BDD and vKITTI with confidence thresholds set to 0.1. The detection confidence scores are labeled on the detected bounding boxes.

Table A.2 Mappings of various dataset labels to MS COCO labels.

KITTI	CAR-CAR; VAN-CAR; TRUCK-TRUCK; PEDESTRIAN-PEDESTRIAN; PERSON-PERSON; CYCLIST-PERSON; TRAM-TRAIN
vKITTI	CAR-CAR; VAN-CAR; TRUCK-TRUCK; TRAM-TRAIN
BDD	PEDESTRIAN-PERSON; RIDER-PERSON; CAR-CAR; TRUCK-TRUCK; BUS-BUS; TRAIN-TRAIN

YOLOv8 is a powerful tool, especially for object detection tasks. Its detection results are very impressive but not always perfect, which leaves rooms for errors when relying on its output to evaluate our model’s performance. Two examples of YOLOv8’s detection on generated and ground-truth BDD frames are illustrated in Figure A.1.

In the ground-truth BDD frame detections, the network has missed detecting the two black SUVs on the right, even though the confidence threshold is set as low as 0.1 during detection.

In the generated BDD frame detections, the network has falsely detected the reflections on the engine hood of the driving vehicle as a car. This is a common mislabeling we have observed across the experiments.

In the last example, we demonstrate that YOLOv8 can generate detections on virtual datasets such as vKITTI.

We have attempted to enhance our detection results across frames by using a tracker in conjunction with the YOLOv8 detector. Specifically, we use the BoT-SORT [140] tracker. However, we’ve observed an increase in the precision but a significant drop in the recall of the detection scores. Thus, we’ve reverted to using only the YOLOv8 detection model to compute the AP metrics.

BDD Result Visualizations: Bounding-Box Predictions and Motion-Controlled Video Generations

In the following section, we showcase a range of BDD generation results produced by our model in different scene scenarios, such as city, urban, highways, busy intersections and at night. Each visualization displays every 6th frame from a 25-frame clip, with the actual video generated at a frame rate of 5 fps. In the leftmost column labels, GT represents ground truth, PB represents predicted bounding-box frames, and PF represents predicted frames.

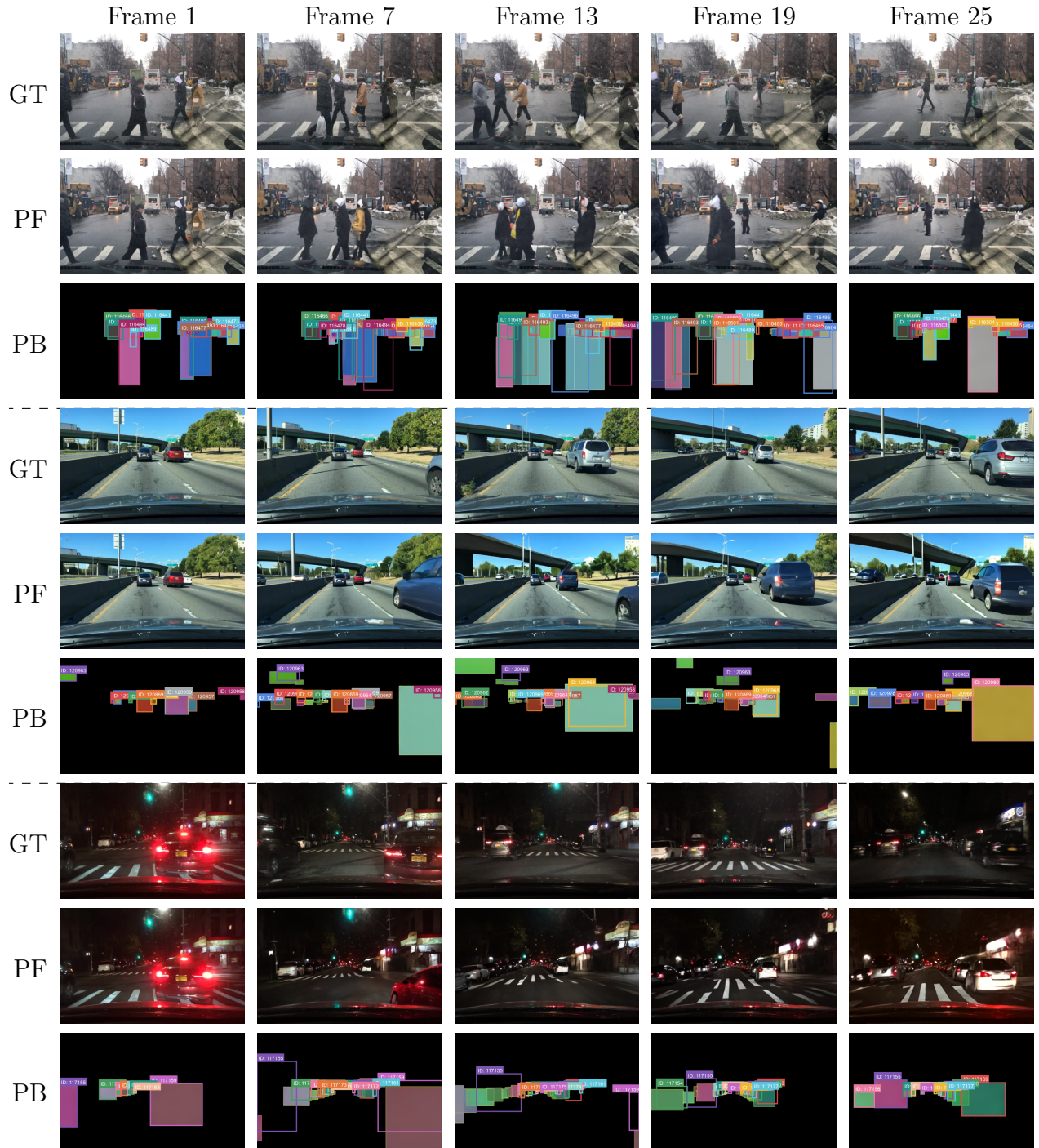


Figure A.2 2D bounding-box frame predictions and motion-controlled video generations for various scenes.



Figure A.3 2D bounding-box frame predictions and motion-controlled video generations for various scenes.

vKITTI Result Visualizations: 3D Bounding-Box Predictions and Motion Controlled Video Generations

In the following section, we showcase a range of vKITTI generation results produced by our model in different scene scenarios. In Figure A.4, we showcase our generations on the vKITTI test split. However, it is worth noting that the vKITTI test split comprises samples from only one type of scene (unseen during training). To provide a broader perspective, in Figure A.5, we present additional results on the vKITTI train split. Each visualization displays every 6th frame from a 25-frame clip, with the actual video generated at a frame rate of 7 fps. In the leftmost column labels, GT represents ground truth, PB represents predicted bounding-box frames, and PF represents predicted frames.

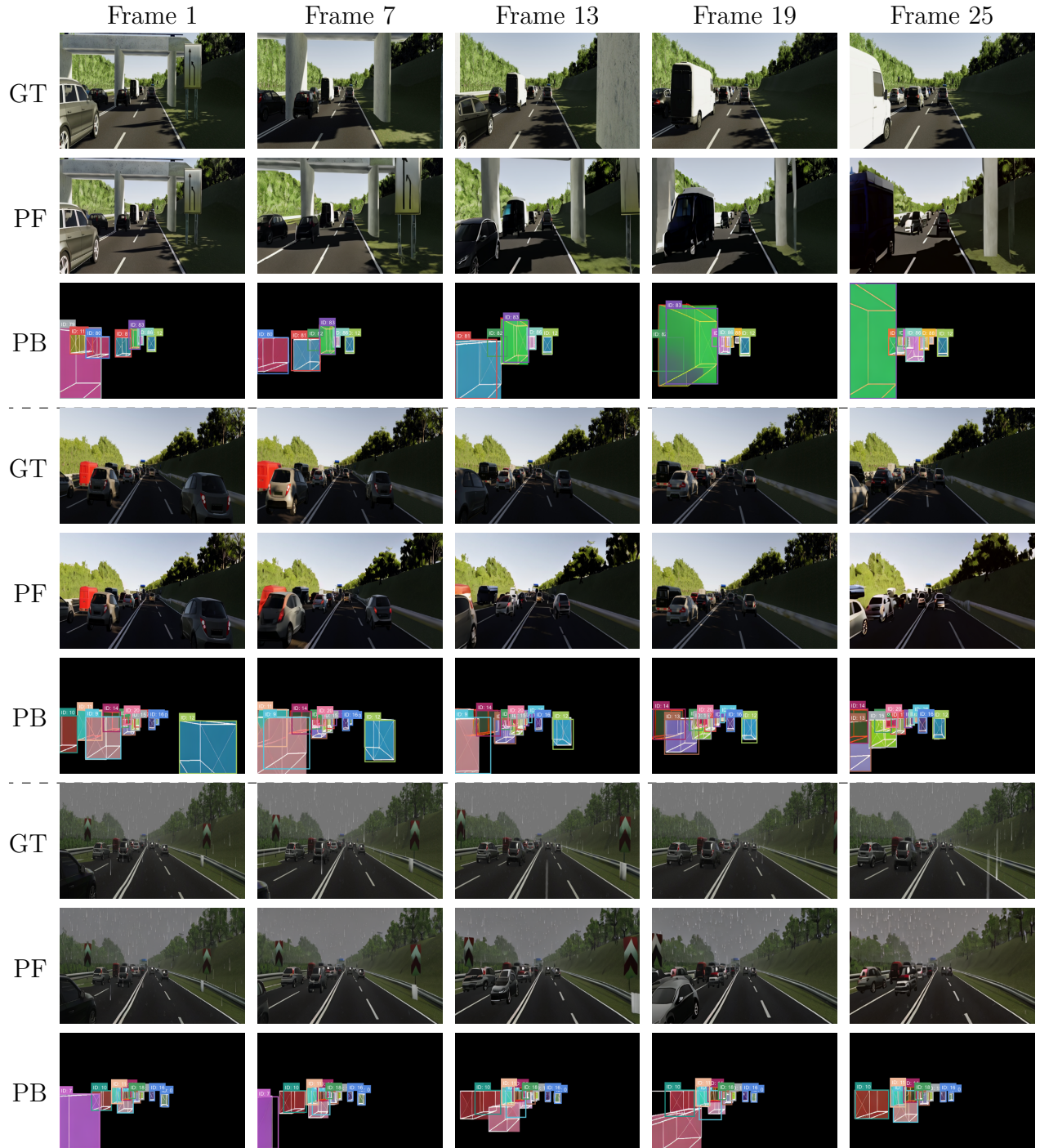


Figure A.4 3D bounding-box predictions and motion-controlled video generations for various scenes on vKITTI test-split.

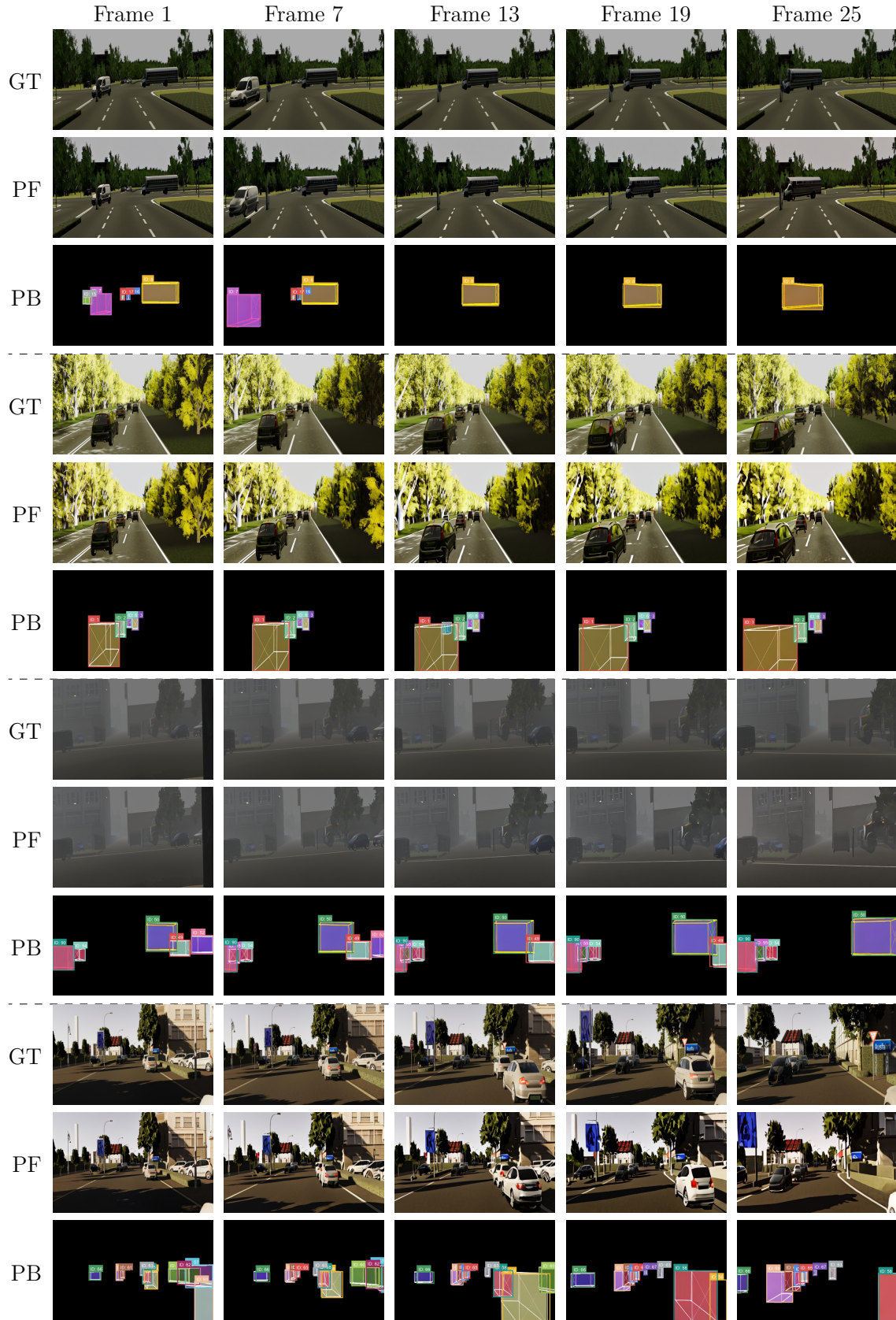


Figure A.5 3D bounding-box predictions and motion-controlled video generations for various scenes on vKITTI train-split.

KITTI Result Visualizations: Bounding-Box Predictions and Motion-Controlled Video Generations

In the following section, we showcase a range of KITTI generation results produced by our model in different scene scenarios, such as urban area, city streets and highway. Each visualization displays every 6th frame from a 25-frame clip, with the actual video generated at a frame rate of 7 fps. In the leftmost column labels, GT represents ground truth, PB represents predicted bounding-box frames, and PF represents predicted frames.

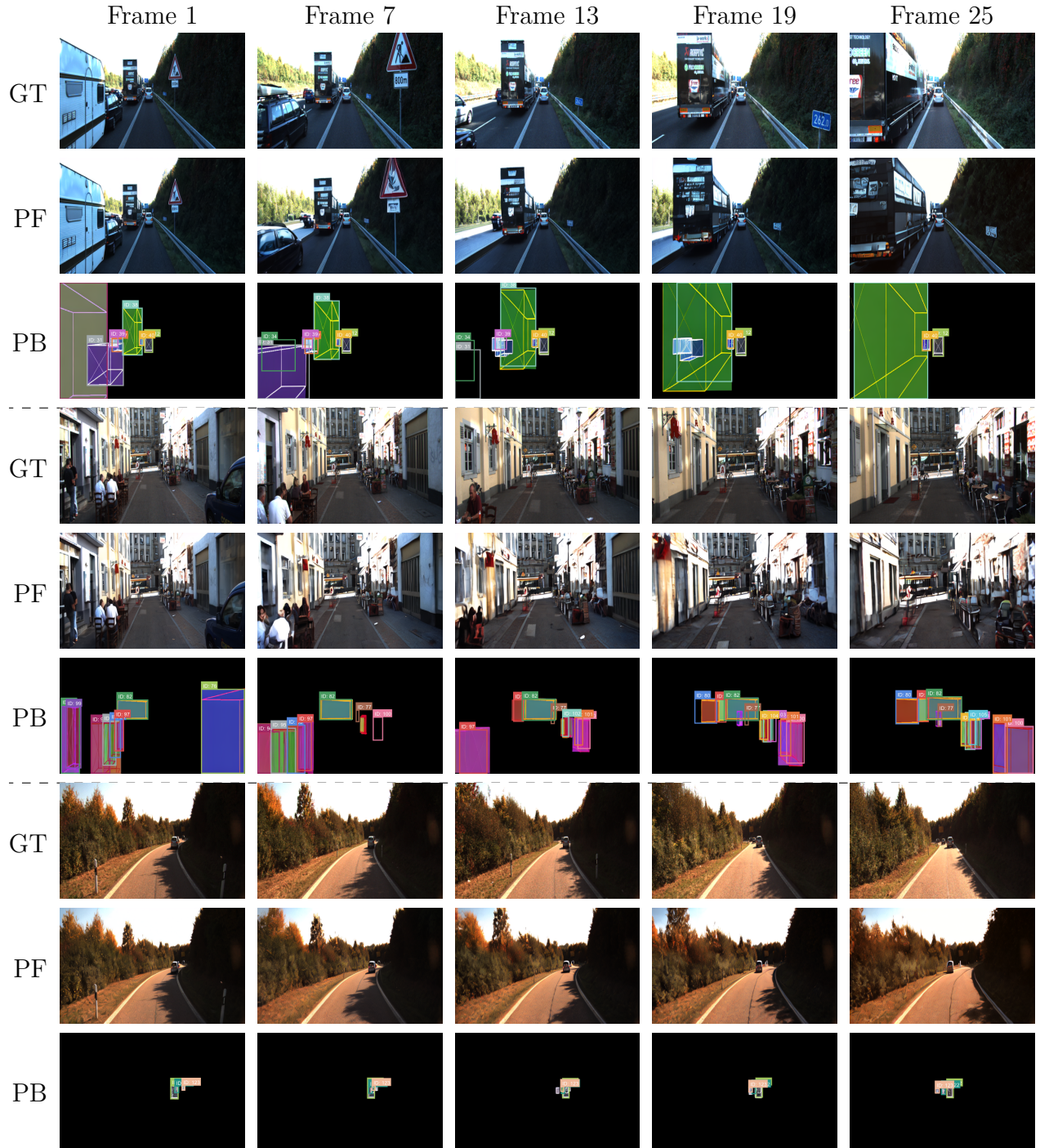


Figure A.6 3D bounding-box predictions and motion-controlled video generations for various scenes on KITTI test-split.

APPENDIX B DS2D: ADDITIONAL RESULTS

Here, we provide detailed Prompt Consistency (PC), Self Consistency (SC), and compatibility metrics from all of our experiments on all model variants.

Generation Prompt: SpecificTable B.1 **Self consistency** metrics on generated results from **Specific Prompt**.

	Greedy Generation						
	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
F	0.94±0.07	0.16±0.37	1.00	0.99±0.02	1.00	0.99±0.02	1.00±0.03
F+BD	0.95±0.06	0.22±0.41	1.00	0.99±0.02	1.00	0.99±0.04	1.00±0.00
M	0.93±0.07	0.18±0.39	1.00	0.99±0.04	1.00	0.98±0.06	1.00±0.00
M+BD	0.93±0.07	0.19±0.39	1.00	0.99±0.04	1.00	0.99±0.03	1.00±0.03
PM	0.92±0.06	0.12±0.32	0.98	0.98±0.07	1.00	1.00±0.01	1.00±0.02
PM+BD	0.90±0.09	0.17±0.37	1.00	1.00±0.03	1.00	1.00±0.01	1.00±0.00

	Sampling Generation						
	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	RID	R. W	Total Area
F	0.95±0.06	0.17±0.38	1.00	0.99±0.02	1.00	0.99±0.02	1.00±0.00
F+BD	0.95±0.06	0.22±0.41	1.00	0.99±0.03	1.00	0.99±0.03	1.00±0.00
M	0.93±0.08	0.16±0.37	1.00	0.99±0.04	1.00	0.98±0.07	1.00±0.00
M+BD	0.93±0.07	0.17±0.38	1.00	0.99±0.04	1.00	0.99±0.04	1.00±0.02
PM	0.91±0.07	0.14±0.35	0.97	0.96±0.09	1.00	1.00±0.02	1.00±0.02
PM+BD	0.90±0.08	0.20±0.40	1.00	0.99±0.06	1.00	1.00±0.00	1.00±0.00

Table B.2 **Prompt consistency** metrics on generated results from **Specific Prompt**.

	Greedy Generation							
	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
F	1.00±0.00	0.95±0.07	0.94±0.07	0.99±0.02	1.00±0.03	1.00±0.00	1.00±0.00	0.99±0.02
F+BD	1.00±0.00	0.96±0.05	0.95±0.06	0.99±0.03	1.00±0.00	1.00±0.00	1.00±0.00	0.99±0.04
M	1.00±0.00	0.94±0.07	0.93±0.07	0.97±0.07	1.00±0.00	1.00±0.00	1.00±0.00	0.97±0.08
M+BD	1.00±0.00	0.95±0.07	0.93±0.07	0.97±0.07	1.00±0.03	1.00±0.00	1.00±0.00	0.97±0.08
PM	1.00±0.00	0.74±0.39	0.92±0.06	0.80±0.17	1.00±0.00	1.00±0.00	1.00±0.00	0.80±0.19
PM+BD	1.00±0.00	0.86±0.25	0.90±0.09	0.83±0.19	1.00±0.00	1.00±0.00	1.00±0.00	0.81±0.19

	Sampling Generation							
	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
F	1.00±0.00	0.95±0.06	0.95±0.06	0.99±0.03	1.00±0.00	1.00±0.00	1.00±0.00	0.99±0.02
F+BD	1.00±0.00	0.96±0.06	0.95±0.06	0.99±0.03	1.00±0.00	1.00±0.00	1.00±0.00	0.99±0.03
M	1.00±0.00	0.95±0.07	0.93±0.08	0.97±0.07	1.00±0.00	1.00±0.00	1.00±0.00	0.96±0.09
M+BD	1.00±0.00	0.95±0.06	0.93±0.07	0.96±0.07	1.00±0.02	1.00±0.00	1.00±0.00	0.97±0.08
PM	1.00±0.00	0.75±0.37	0.91±0.07	0.78±0.17	1.00±0.01	1.00±0.00	1.00±0.00	0.77±0.20
PM+BD	1.00±0.00	0.86±0.26	0.90±0.08	0.79±0.19	1.00±0.00	1.00±0.00	1.00±0.00	0.79±0.19

Generation Prompt: Total AreaTable B.3 **Self consistency** metrics on generated results from **Total Area Prompt**.

	Greedy Generation						
	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
F	0.79±0.13	0.07±0.26	0.96	1.00±0.01	1.00	1.00±0.01	0.97±0.07
F+BD	0.83±0.09	0.19±0.39	0.97	1.00±0.02	1.00	0.99±0.03	0.97±0.09
M	0.92±0.06	0.09±0.28	1.00	1.00±0.01	1.00	1.00±0.01	0.99±0.02
M+BD	0.91±0.06	0.14±0.35	1.00	1.00±0.01	1.00	1.00±0.01	1.00±0.02
PM	0.93±0.05	0.14±0.35	0.95	1.00±0.01	1.00	1.00±0.01	0.99±0.04
PM+BD	0.90±0.07	0.13±0.34	0.98	1.00±0.01	1.00	1.00±0.01	0.98±0.12

	Sampling Generation						
	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
F	0.78±0.13	0.14±0.34	0.96	0.99±0.02	1.00	0.99±0.02	0.98±0.06
F+BD	0.81±0.10	0.26±0.44	0.96	0.99±0.02	1.00	0.99±0.02	0.96±0.15
M	0.90±0.06	0.14±0.35	1.00	1.00±0.01	1.00	1.00±0.01	1.00±0.03
M+BD	0.90±0.06	0.15±0.35	1.00	1.00±0.01	1.00	1.00±0.01	0.99±0.03
PM	0.92±0.06	0.18±0.38	0.97	1.00±0.01	1.00	1.00±0.01	0.99±0.03
PM+BD	0.88±0.07	0.15±0.35	0.96	1.00±0.01	1.00	1.00±0.01	0.96±0.18

Table B.4 **Prompt consistency** metrics on generated results from **Total Area Prompt**.

	Greedy Generation							
	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
F	1.00±0.00	0.83±0.14	-	-	-	-	-	-
F+BD	1.00±0.00	0.87±0.12	-	-	-	-	-	-
M	1.00±0.00	0.94±0.06	-	-	-	-	-	-
M+BD	1.00±0.00	0.94±0.05	-	-	-	-	-	-
PM	1.00±0.00	0.95±0.06	-	-	-	-	-	-
PM+BD	1.00±0.00	0.91±0.13	-	-	-	-	-	-

	Sampling Generation							
	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
F	1.00±0.00	0.82±0.15	-	-	-	-	-	-
F+BD	1.00±0.00	0.84±0.16	-	-	-	-	-	-
M	1.00±0.00	0.93±0.06	-	-	-	-	-	-
M+BD	1.00±0.00	0.93±0.06	-	-	-	-	-	-
PM	1.00±0.00	0.94±0.05	-	-	-	-	-	-
PM+BD	1.00±0.00	0.88±0.18	-	-	-	-	-	-

Generation Prompt: Partial Room Area

Table B.5 **SC** metrics on generated results from **Partial Room Area Prompt**

Greedy Generation							
Self Consistency Metrics							
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
F	0.81±0.11	0.03±0.17	0.69	1.00±0.01	1.00	1.00±0.02	0.86±0.21
F+BD	0.79±0.10	0.17±0.37	0.76	1.00±0.02	1.00	0.99±0.03	0.92±0.15
M	0.90±0.07	0.11±0.31	0.94	0.99±0.02	1.00	1.00±0.01	0.97±0.10
M+BD	0.90±0.06	0.13±0.34	0.99	1.00±0.01	1.00	1.00±0.01	0.99±0.04
PM	0.92±0.06	0.15±0.35	0.96	1.00±0.01	1.00	1.00±0.01	0.97±0.07
PM+BD	0.90±0.07	0.14±0.34	1.00	1.00±0.01	1.00	1.00±0.01	0.99±0.03

Sampling Generation							
Self Consistency Metrics							
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
F	0.79±0.13	0.06±0.25	0.75	0.99±0.02	1.00	0.99±0.03	0.87±0.22
F+BD	0.78±0.11	0.23±0.42	0.81	0.99±0.02	1.00	0.99±0.02	0.94±0.13
M	0.89±0.07	0.14±0.34	0.94	1.00±0.01	1.00	1.00±0.02	0.97±0.11
M+BD	0.89±0.07	0.14±0.35	0.99	1.00±0.01	1.00	1.00±0.01	0.99±0.03
PM	0.91±0.07	0.18±0.39	0.98	1.00±0.01	1.00	1.00±0.01	0.98±0.04
PM+BD	0.88±0.07	0.17±0.38	1.00	1.00±0.01	1.00	1.00±0.01	0.99±0.04

Table B.6 **PC** metrics on generated results from **Partial Room Area Prompt**

Greedy Generation								
Prompt Consistency Metrics								
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
F	1.00±0.00	0.48±0.40	0.82±0.14	-	0.99±0.09	1.00±0.02	0.99±0.07	-
F+BD	1.00±0.00	0.74±0.21	0.79±0.14	-	0.97±0.15	1.00±0.00	0.99±0.06	-
M	1.00±0.00	0.91±0.10	0.80±0.35	-	0.99±0.07	0.89±0.27	0.99±0.09	-
M+BD	1.00±0.00	0.93±0.06	0.87±0.13	-	1.00±0.05	0.99±0.11	1.00±0.00	-
PM	1.00±0.00	0.93±0.07	0.91±0.11	-	0.99±0.05	1.00±0.00	1.00±0.04	-
PM+BD	1.00±0.00	0.92±0.07	0.89±0.10	-	1.00±0.02	1.00±0.00	1.00±0.01	-

Sampling Generation								
Prompt Consistency Metrics								
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
F	1.00±0.02	0.50±0.39	0.80±0.15	-	0.99±0.07	1.00±0.00	1.00±0.06	-
F+BD	1.00±0.00	0.73±0.25	0.75±0.23	-	0.96±0.16	1.00±0.02	0.99±0.07	-
M	1.00±0.00	0.90±0.12	0.80±0.26	-	0.99±0.07	0.88±0.29	0.99±0.07	-
M+BD	1.00±0.00	0.93±0.07	0.87±0.14	-	0.99±0.06	0.98±0.15	1.00±0.00	-
PM	1.00±0.00	0.94±0.06	0.90±0.11	-	1.00±0.03	1.00±0.00	1.00±0.01	-
PM+BD	1.00±0.00	0.90±0.08	0.87±0.13	-	1.00±0.02	1.00±0.00	1.00±0.01	-

Generation Prompt: All Room Area

Table B.7 **SC** metrics on generated results from **All Room Area Prompt**.

	Greedy Generation						
	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
F	0.82±0.10	0.16±0.37	1.00	1.00±0.02	1.00	1.00±0.01	0.93±0.17
F+BD	0.76±0.11	0.25±0.43	1.00	0.99±0.02	1.00	0.99±0.02	0.90±0.20
M	0.90±0.07	0.12±0.33	1.00	0.99±0.02	1.00	0.99±0.01	0.99±0.03
M+BD	0.91±0.06	0.14±0.35	1.00	1.00±0.01	1.00	1.00±0.01	0.99±0.02
PM	0.91±0.06	0.14±0.35	1.00	1.00±0.01	1.00	1.00±0.01	0.97±0.09
PM+BD	0.90±0.07	0.19±0.39	1.00	1.00±0.01	1.00	1.00±0.01	0.99±0.05

	Sampling Generation						
	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
F	0.80±0.10	0.18±0.39	0.99	0.99±0.02	1.00	0.99±0.04	0.95±0.14
F+BD	0.76±0.11	0.31±0.46	1.00	0.99±0.02	1.00	0.99±0.03	0.93±0.15
M	0.89±0.07	0.17±0.37	1.00	1.00±0.01	1.00	0.99±0.02	0.99±0.03
M+BD	0.90±0.07	0.15±0.36	1.00	1.00±0.01	1.00	1.00±0.01	0.99±0.04
PM	0.91±0.07	0.18±0.38	1.00	1.00±0.01	1.00	1.00±0.01	0.96±0.10
PM+BD	0.88±0.07	0.19±0.40	1.00	1.00±0.01	1.00	1.00±0.01	1.00±0.00

Table B.8 **PC** metrics on generated results from **All Room Area Prompt**.

	Greedy Generation							
	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
F	1.00±0.00	-	0.81±0.10	-	1.00±0.06	1.00±0.00	1.00±0.06	-
F+BD	1.00±0.00	-	0.76±0.11	-	1.00±0.00	1.00±0.00	1.00±0.00	-
M	1.00±0.00	-	0.90±0.07	-	1.00±0.00	1.00±0.00	1.00±0.00	-
M+BD	1.00±0.00	-	0.91±0.06	-	1.00±0.01	1.00±0.00	1.00±0.00	-
PM	1.00±0.00	-	0.91±0.06	-	1.00±0.00	1.00±0.00	1.00±0.00	-
PM+BD	1.00±0.00	-	0.90±0.07	-	1.00±0.00	1.00±0.00	1.00±0.00	-

	Sampling Generation							
	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
F	1.00±0.00	-	0.79±0.10	-	0.99±0.09	1.00±0.00	0.99±0.09	-
F+BD	1.00±0.00	-	0.76±0.11	-	1.00±0.00	1.00±0.00	1.00±0.00	-
M	1.00±0.00	-	0.89±0.07	-	1.00±0.00	1.00±0.00	1.00±0.00	-
M+BD	1.00±0.00	-	0.90±0.07	-	1.00±0.04	1.00±0.00	1.00±0.00	-
PM	1.00±0.00	-	0.91±0.07	-	1.00±0.01	1.00±0.00	1.00±0.01	-
PM+BD	1.00±0.00	-	0.88±0.07	-	1.00±0.02	1.00±0.00	1.00±0.00	-

Compatibility Metrics on ProcTHOR

We further run compatibility evaluations on all of the Bubble Diagram model variants to test how well bubble diagrams are followed. The results are presented here:

Table B.9 Compatibility ($\downarrow$) score on ProcTHOR experiments. The best score in each category (by number of rooms in a floor plan) is bolded, and the 2nd best score has an asterisk.

Model	Generation	Number of Rooms				
Variant	Prompt	3	4	5	6	7
F+BD	Specific	0.15 $\pm$ 0.41	0.33 $\pm$ 0.58	1.35 $\pm$ 1.34	2.24 $\pm$ 1.76	4.23 $\pm$ 1.89
	Total Area	0.19 $\pm$ 0.40	0.62 $\pm$ 0.78	3.48 $\pm$ 1.83	3.41 $\pm$ 1.50	6.23 $\pm$ 2.20
	Partial Room Area	3.80 $\pm$ 1.20	5.73 $\pm$ 2.03	8.19 $\pm$ 2.72	10.15 $\pm$ 2.88	11.42 $\pm$ 4.52
	All Room Area	5.91 $\pm$ 0.29	9.75 $\pm$ 0.54	13.79 $\pm$ 1.15	18.09 $\pm$ 1.71	22.35 $\pm$ 2.24
M+BD	Specific	0.16 $\pm$ 0.45	0.47 $\pm$ 0.67	2.06 $\pm$ 1.33	2.85 $\pm$ 1.79	5.23 $\pm$ 2.14
	Total Area	0.20 $\pm$ 0.40	0.54 $\pm$ 0.73	2.40 $\pm$ 1.03	3.09 $\pm$ 2.07	6.26 $\pm$ 2.03
	Partial Room Area	3.85 $\pm$ 1.11	6.35 $\pm$ 1.95	8.44 $\pm$ 2.48	11.56 $\pm$ 3.42	14.13 $\pm$ 3.90
	All Room Area	5.91 $\pm$ 0.29	9.62 $\pm$ 0.56	12.90 $\pm$ 1.27	17.48 $\pm$ 1.82	22.13 $\pm$ 1.94
PM+BD	Specific	0.15 $\pm$ 0.39	0.58 $\pm$ 0.96	2.42 $\pm$ 1.33	2.97 $\pm$ 1.62	5.97 $\pm$ 1.25
	Total Area	0.34 $\pm$ 0.93	0.46 $\pm$ 0.84	2.08 $\pm$ 1.29	2.65 $\pm$ 1.87	5.84 $\pm$ 1.61
	Partial Room Area	3.87 $\pm$ 1.13	6.42 $\pm$ 1.97	8.99 $\pm$ 2.67	12.06 $\pm$ 3.05	13.68 $\pm$ 3.71
	All Room Area	5.88 $\pm$ 0.33	9.77 $\pm$ 0.49	13.17 $\pm$ 1.23	18.56 $\pm$ 1.58	21.97 $\pm$ 2.36

Self and Prompt Consistency Metrics on RPLAN

This section contains the full self- and prompt-consistency metrics evaluated on the 4 models (Section 5.6.3) trained on the converted RPLAN dataset.

Similar to the ProcTHOR-trained model experiments, we run the 4 different generation prompts on each of the RPLAN-trained model variants, resulting in 16 different experiments. They are organized by generation prompt and shown here.

Generation Prompt: Specific

Table B.10 **Self consistency** metrics on RPLAN generations using **Specific** generation prompts.

	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
5-Room	0.89±0.31	0.34±0.47	0.40	1.00±0.01	1.00	0.98±0.03	0.92±0.08
6-Room	0.93±0.05	0.36±0.48	0.99	0.99±0.01	1.00	0.97±0.02	0.98±0.06
7-Room	0.93±0.04	0.42±0.49	0.97	0.99±0.01	1.00	0.98±0.02	0.98±0.05
8-Room	0.93±0.07	0.62±0.49	0.89	0.99±0.01	1.00	0.98±0.02	0.95±0.08

Table B.11 **Prompt consistency** metrics on RPLAN generations using **Specific** generation prompts.

	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
5-Room	1.00±0.00	0.92±0.06	0.93±0.09	0.94±0.11	1.00±0.00	1.00±0.00	1.00±0.00	0.95±0.09
6-Room	1.00±0.00	0.94±0.07	0.94±0.06	0.92±0.23	1.00±0.05	1.00±0.00	1.00±0.00	0.94±0.14
7-Room	1.00±0.00	0.94±0.06	0.94±0.06	0.93±0.17	1.00±0.02	1.00±0.00	1.00±0.00	0.94±0.29
8-Room	1.00±0.00	0.92±0.08	0.94±0.07	0.95±0.10	0.99±0.06	1.00±0.00	1.00±0.00	0.96±0.09

Generation Prompt: Total Area

Table B.12 **Self consistency** metrics on RPLAN generations using **Total Area** generation prompts.

	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
5-Room	0.89±0.41	0.31±0.46	0.19	1.00±0.01	1.00	0.98±0.03	0.95±0.06
6-Room	0.93±0.05	0.32±0.47	1.00	0.99±0.01	1.00	0.98±0.02	0.99±0.03
7-Room	0.94±0.03	0.38±0.48	1.00	0.99±0.01	1.00	0.98±0.02	0.99±0.02
8-Room	0.82±3.50	0.53±0.50	0.89	0.99±0.01	1.00	0.98±0.02	0.97±0.07

Table B.13 **Prompt consistency** metrics on RPLAN generations using **Total Area** generation prompts.

	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
5-Room	-	0.92±0.07	-	-	-	-	-	-
6-Room	-	0.94±0.06	-	-	-	-	-	-
7-Room	-	0.94±0.06	-	-	-	-	-	-
8-Room	-	0.93±0.07	-	-	-	-	-	-

Generation Prompt: Partial Room Area

Table B.14 **Self consistency** metrics on RPLAN generations using **Partial Room Area** generation prompts.

	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
5-Room	0.77±1.93	0.25±0.44	0.49	1.00±0.01	1.00	0.97±0.10	0.95±0.08
6-Room	0.93±0.07	0.34±0.47	0.98	0.99±0.04	1.00	0.97±0.02	0.97±0.08
7-Room	0.93±0.04	0.40±0.49	0.96	0.99±0.02	1.00	0.98±0.02	0.96±0.07
8-Room	0.91±0.34	0.54±0.50	0.73	0.99±0.02	1.00	0.97±0.02	0.93±0.11

Table B.15 **Prompt consistency** metrics on RPLAN generations using **Partial Room Area** generation prompts.

	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
5-Room	-	0.92±0.08	0.91±0.14	-	1.00±0.05	1.00±0.04	1.00±0.00	-
6-Room	-	0.93±0.08	0.86±0.72	-	0.99±0.06	0.99±0.07	1.00±0.02	-
7-Room	-	0.91±0.09	0.88±0.19	-	1.00±0.03	0.99±0.06	1.00±0.01	-
8-Room	-	0.89±0.11	0.73±0.48	-	0.98±0.09	0.94±0.13	0.99±0.05	-

Generation Prompt: All Room Area

Table B.16 **Self consistency** metrics on RPLAN generations using **All Room Area** generation prompts.

	Self Consistency Metrics						
	P. Area	Overlap	R. Count	R. H	ID	R. W	Total Area
5-Room	0.90±0.16	0.37±0.48	0.97	1.00±0.01	1.00	0.98±0.02	0.98±0.05
6-Room	0.93±0.06	0.31±0.46	1.00	0.99±0.01	1.00	0.97±0.03	0.95±0.04
7-Room	0.92±0.15	0.38±0.49	0.93	0.99±0.02	1.00	0.98±0.02	0.98±0.03
8-Room	0.93±0.10	0.61±0.49	0.96	0.99±0.02	1.00	0.98±0.02	0.91±0.07

Table B.17 **Prompt consistency** metrics on RPLAN generations using **All Room Area** generation prompts.

	Prompt Consistency Metrics							
	Num. R.	Total Area	R. Area	R. H	ID	IDvsType	Type	R. W
5-Room	-	-	0.92±0.04	-	1.00±0.01	1.00±0.00	1.00±0.00	-
6-Room	-	-	0.93±0.05	-	1.00±0.02	1.00±0.00	1.00±0.00	-
7-Room	-	-	0.93±0.07	-	1.00±0.02	1.00±0.00	1.00±0.00	-
8-Room	-	-	0.93±0.05	-	0.99±0.04	1.00±0.00	1.00±0.00	-