Titre: Title:	Controllable Realistic Simulation and Video Generation for Autonomous Vehicles in Safety-Critical Scenarios
Auteur: Author:	Anthony Gosselin
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gosselin, A. (2025). Controllable Realistic Simulation and Video Generation for Autonomous Vehicles in Safety-Critical Scenarios [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/68994/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/68994/
Directeurs de recherche: Advisors:	Christopher J. Pal
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Controllable Realistic Simulation and Video Generation for Autonomous Vehicles in Safety-Critical Scenarios

ANTHONY GOSSELIN
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Septembre 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Controllable Realistic Simulation and Video Generation for Autonomous Vehicles in Safety-Critical Scenarios

présenté par **Anthony GOSSELIN**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Giovanni BELTRAME, président
Christopher J. PAL, membre et directeur de recherche
Nicolas SAUNIER, membre

DEDICATION

To my family and friends,
for making the peaks joyous and the valleys worth crossing...

ACKNOWLEDGEMENTS

Not only have I been fortunate to stand on the shoulders of giants, I have also had the privilege of working alongside some of them. My heartfelt thanks go to Roger Girgis, Luke Rowe, and Olga Luo—colleagues, co-authors, and friends—whose mentorship and guidance have shaped me into a better researcher.

I am equally grateful to my supervisor, Christopher Pal, for granting me such a formative opportunity, and to all those who generously shared their time, insight, and expertise, including Alexia Jolicœur-Martineau, Florian Golemo, Luis Lara, Derek Nowrouzezahrai, Liam Paull, among others.

Finally, thank you to my friends and to everyone—near and far—at Mila who helped make this journey as rewarding and enjoyable as it has been. And for allowing me to fulfill my dream of making two cars go boom on my computer screen.

RÉSUMÉ

Assurer la sécurité des véhicules autonomes (VA) nécessite de les tester sur un large éventail de scénarios, y compris des événements rares et dangereux tels que les collisions. La collecte de telles données dans le monde réel est dangereuse, coûteuse et intrinsèquement limitée en diversité. Les simulateurs traditionnels offrent contrôle et réalisme physique, mais requièrent une création manuelle importante d'éléments visuels et peinent encore à reproduire la diversité et le réalisme visuel des images réelles de conduite. Cette thèse explore la simulation fondée sur les données à l'aide de modèles de diffusion vidéo génératifs, capables de synthétiser des vidéos de conduite photoréalistes et temporellement cohérentes avec un effort humain minimal, offrant ainsi une alternative évolutive et contrôlable pour les tests de sécurité des VA.

Pour commencer, on présente la simulation contrôlable en vue de dessus (*bird's-eye view* ou BEV) pour la modélisation de comportements de véhicules à haut niveau. Deux contributions antérieures, CtRL-Sim et Scenario Dreamer, permettent la génération de comportements de conduite multi-agents et de réseaux routiers complets en BEV. Bien que puissants pour la création de scénarios, les environnements BEV manquent le niveau de détail visuel nécessaire pour tester les systèmes de perception ou produire des vidéos réalistes d'accidents. Pour combler ces lacunes, la thèse présente la méthode BEV2POV, qui traduit des simulations BEV structurées en vidéos du point de vue du conducteur. Au cœur de ce système se trouve Ctrl-V, un modèle de diffusion vidéo contrôlable capable de générer des scènes de conduite structurées à partir de trajectoires de boîtes englobantes. Sur cette base, la contribution principale, Ctrl-Crash, étend l'approche pour générer des scénarios d'accidents réalistes et contrôlables. Ctrl-Crash prend en entrée une scène initiale, les trajectoires des agents et un type d'accident, puis produit des vidéos haute fidélité alignées à la fois avec les conditions d'entrée et avec une dynamique physique plausible. Les résultats montrent que Ctrl-Crash surpasse systématiquement les modèles de génération d'accidents par diffusion existants et prend en charge la génération de scénarios contrefactuels, produisant plusieurs issues plausibles à partir des mêmes conditions initiales.

La thèse se conclut par une discussion des limitations et des perspectives de recherche futures dans le domaine de la simulation neuronale pour les VA. Dans leur ensemble, ces contributions font progresser le développement d'outils de simulation réalistes, contrôlables et évolutifs pour la recherche sur les scénarios critiques pour la sécurité des véhicules autonomes.

ABSTRACT

Ensuring the safety of autonomous vehicles (AVs) requires testing across a wide range of scenarios, including rare and dangerous events such as collisions. Collecting such data in the real world is unsafe, costly, and inherently limited in diversity. Traditional physics-based simulators provide control and physical realism but require extensive manual asset creation and still fall short of matching the visual diversity and realism of real-world driving footage. This thesis explores data-driven simulation using generative video diffusion models, which can synthesize photorealistic, temporally consistent driving videos with minimal human effort, offering a scalable and controllable alternative for AV safety testing.

The work begins with controllable bird's-eye view (BEV) simulation for high-level traffic behavior modeling. Two prior contributions, CtRL-Sim and Scenario Dreamer, enable the generation of multi-agent driving behaviors and entire traffic environments in BEV. While powerful for scenario creation, BEV lacks the visual detail needed to test perception systems or produce realistic crash footage. To bridge this gap, the thesis introduces the BEV2POV framework, which translates structured BEV simulations into driver's-view videos. At its core is Ctrl-V, a controllable video diffusion model capable of generating structured driving scenes from bounding box trajectories. Building on this, the main contribution, Ctrl-Crash, extends the approach to generate realistic and controllable crash scenarios. Ctrl-Crash accepts an initial scene, agent trajectories, and a semantic crash type, producing high-fidelity videos aligned with both the input conditions and plausible physical dynamics. Results show that Ctrl-Crash consistently outperforms prior diffusion-based crash generation models and supports counterfactual scenario generation, producing multiple plausible outcomes from the same starting conditions.

The thesis concludes by discussing limitations and outlining future research directions in the field of AV neural simulation. Together, these contributions advance the development of realistic, controllable, and scalable simulation tools for safety-critical AV research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

AV	Autonomous Vehicle
BEV	Bird's-Eye View
CFG	Classifier-Free Guidance
DDPM	Denoising Diffusion Probabilistic Model
DDIM	Denoising Diffusion Implicit Model
FFT	Fast Fourier Transform
FVD	Fréchet Video Distance
GT	Ground-Truth
IoU	Intersection over Union
JEPA	Joint Embedding Predictive Architecture
JEDi	JEPA Embedding Distance
LPIPS	Learned Perceptual Image Patch Similarity
MMD	Maximum Mean Discrepancy
OOD	Out-of-Distribution
POV	Point of View
PSNR	Peak Signal-to-Noise Ratio
SOTA	State-of-the-Art
SSIM	Structural Similarity Index
SVD	Stable Video Diffusion
VAE	Variational Autoencoder

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Every time an autonomous vehicle (AV) drives safely down the road, it's the result of countless of hours of training and testing in both the real and simulated world. But while it's easy to collect data for normal driving, the situations that matter most—those that involve split-second decisions, unpredictable agents, or life-threatening risks—are the hardest to capture. These are the safety-critical scenarios: crashes, near-misses, emergency braking, vehicles cutting in without warning. Imagine trying to teach a self-driving car how to avoid a collision with a reckless driver who runs a red light at high speed. Such situations are rare in the real world, but that rarity is precisely the problem. Real-world data alone is not enough to prepare AVs for the long tail of edge cases that matter most. To truly evaluate and improve safety, we need the ability to simulate dangerous events in a controllable and realistic way, so we can test how AVs respond before lives are on the line. This thesis focuses on building such a simulation capability, with an emphasis on one of the most difficult and underrepresented scenarios of all: car crashes.

What does it mean to simulate a crash in a way that is both controllable and realistic? Broadly, controllability refers to the ability to define, manipulate, and intervene in the elements of a simulated scenario. A controllable simulator allows users to specify what happens, when, and how—whether by adjusting high-level parameters such as traffic density, or by precisely defining low-level behaviors of individual agents like exact motion and trajectories in time. Realism, on the other hand, speaks to the plausibility and fidelity of the scenario. A realistic simulation must adhere to the rules of physics, reflect social driving conventions, and, when necessary, exhibit visually accurate phenomena such as occlusion, motion blur, or damage.

In the context of autonomous vehicle simulation, these two properties are not merely desirable, they are essential. Controllability allows for targeted testing of safety-critical edge cases, while realism ensures that these tests are meaningful and predictive of real-world performance. Throughout this thesis, these concepts will be explored across different forms of representation. In bird's-eye view (BEV) settings, controllability relates to manipulating structured agent trajectories and scene layouts, while realism is measured by the plausibility of multi-agent behaviors in traffic. In driver's point of view (POV) video generation, controllability takes the form of fine-grained object-level conditioning for motion and appearance, and realism is judged by the visual fidelity of the synthesized driving videos.

These interpretations of control and realism form the conceptual backbone of this work. The

following chapters draw on several publications to which the author contributed (see Appendix A) to develop techniques for generating safety-critical scenarios under these principles, beginning with controllable behavior modeling in BEV and culminating in the generation of realistic crash videos from the driver's perspective. This thesis draws on several publications to which the author contributed

We first lay the foundation for this work by providing a theoretical and mathematical background of key technical concepts in Chapter 2. Then, to develop the ideas described above, this thesis is structured around a progression of techniques that enable increasingly realistic and controllable simulation of safety-critical scenarios. Chapter 3 builds on prior work co-authored by the author, including CtRL-Sim and Scenario Dreamer, which focus on simulating behavior in a top-down view. These methods introduce mechanisms for generating agent trajectories and scene layouts with interpretable control, setting the foundation for structured scenario generation. Chapter 4 turns to Ctrl-V, another co-authored work, which addresses video generation in the egocentric view using bounding box-conditioned diffusion models. Together, these two threads (structured BEV simulation and visually grounded video synthesis) highlight complementary strengths for modeling complex driving scenes. To bridge these representations, Chapter 5 introduces a lightweight geometric framework called BEV2POV, which transforms BEV trajectories into POV video conditioning inputs. This method provides a proof of concept and supporting experiments that demonstrate how structured behavioral models and visual generation frameworks can be combined. This leads to the central contribution of this thesis in Chapter 6: a controllable video generation model tailored for crash simulation, designed to overcome the limitations of general-purpose models in handling out-of-distribution safety events. This model, along with the crash-focused dataset and evaluation methodology that support it, are presented here.

CHAPTER 2 BACKGROUND

2.1 Car Crash Simulation and AV Safety.

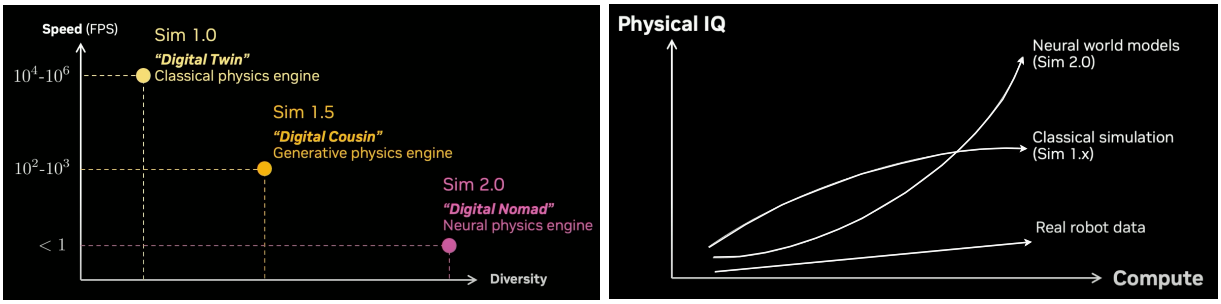
Simulation plays a critical role in the development and validation of autonomous vehicles, offering a scalable and safe means of testing driving systems under diverse and challenging conditions. Because dangerous edge cases, such as collisions, near-misses, and extreme weather are rare in real-world datasets, relying solely on on-road testing is impractical and unethical. Simulators enable the controlled reproduction of such events, allowing AV systems to be evaluated under safety-critical situations that would be unethical or unsafe to stage in the real world. Beyond evaluation, simulation is also key to training perception, prediction, and planning models, especially in low-data regimes. Synthetic sensor data can be used to augment real-world datasets and expose AV systems to novel or adversarial situations. This accelerates development and helps improve generalization across driving domains.

However, the utility of simulation is ultimately bounded by its realism and diversity. A simulator that fails to reflect the complex dynamics and variability of real-world driving may lead to overfitting, poor generalization, or unsafe deployment. As such, there is growing interest in enhancing the fidelity, diversity, and controllability of simulated environments, particularly through data-driven techniques such as generative models. The remainder of this section explores the evolution of AV simulation from traditional rule-based engines to hybrid and fully generative approaches, highlighting the limitations of each and motivating the direction taken in this thesis.

2.1.1 From Classical Simulation to Generative Simulation

The development of embodied AI systems, such as AVs and robotics, relies heavily on simulation. It enables scalable data collection, policy training, and safety validation; tasks that are otherwise constrained by real-world limitations such as cost, danger, and environment variability. Without access to a simulator, researchers are limited to collecting real-world data manually, such as through fleet driving or teleoperated robots. This not only incurs significant operational costs, but also restricts the types of scenarios that can be encountered—particularly rare or dangerous edge cases that are critical for robust decision-making.

To address these limitations, simulation platforms aim to provide virtual environments where agents can be trained safely and efficiently. A key advantage is the ability to create digital twins, that is, virtual analogs of the real world that can be parallelized across GPUs and



(a) **Simulation Methods** (b) **Embodied Scaling Law**

Figure 2.1 **Progression and scaling of simulation methods. (Left)**: Illustrates the speed/diversity trade-off of different simulations paradigms. Generative physics engine are hybrid solutions intersecting classic physic simulators and generative methods for augmentation. Neural physics engines are data-driven methods built on neural networks, such as diffusion models or other generative models. **(Right)**: Projected scaling of robotic simulation where "Physical IQ" refers generally to capabilities of physical robotic systems and "Compute" refers to the required data and computation time. Figures source: Jim Fan, "The Physical Turing Test" [1]

modified freely. This unlocks powerful strategies such as domain randomization, where visual appearance, environmental layouts, lighting, and agent behaviors are varied randomly across simulation rollouts. The goal is to expose learning agents to a wide diversity of scenarios, thereby improving generalization to the real world. If the simulation is sufficiently diverse and realistic, the hope is that real-world deployment becomes just another distribution shift the model has already learned to handle; a key objective in sim2real research.

However, this ideal is difficult to realize in practice. Traditional simulators like CARLA [2], AirSim [3], and MuJoCo [4] offer high-fidelity physics and sensor modeling, but require extensive manual effort to scale. Scene assets must be manually designed, environments handcrafted, and behavior policies scripted—making diversity expensive. While these tools excel at structured testing and physical grounding, they often fall short in realism and variety, particularly when simulating open-world scenes or rare adversarial events.

To improve diversity without compromising structure, recent efforts have introduced hybrid approaches that integrate generative models into classical simulators [5, 6]. For example, synthetic textures, 3D objects, or scene layouts can be generated using diffusion models, GANs, or other generative pipelines before being rendered by a traditional graphics engine. This allows simulation content to be diversified more efficiently while maintaining physical plausibility. Still, even the most advanced graphics pipelines—though steadily improving over decades—struggle with visually complex phenomena, for example, semi-transparency,

reflectance, and fine-grained object textures.

In contrast, generative video models—particularly diffusion-based models—have made dramatic leaps in capability within just the past few years. Recent advancements [7–10] have shown these models can synthesize high-fidelity, temporally coherent video with minimal supervision, and scale gracefully with data and compute. This has led to a growing paradigm shift: instead of using generative models to augment simulation, use them to replace it entirely. In this emerging view, a diffusion model trained on real-world video can serve as a data-driven simulator, directly producing plausible sensory observations under high-level control. These methods are often referred to as "world models" or "neural simulators" (See Figure 2.1, left).

This trend, captured in what Jim Fan [1] refers to as the "Embodied AI Scaling Law" (see Figure 2.1, right), suggests that generative models trained on large-scale internet video can outperform hand-engineered simulation pipelines, particularly for tasks involving visual realism, complex dynamics, or rare events. By conditioning these models with structured inputs—such as bounding boxes, text descriptions, or semantic maps—researchers can steer them toward specific scenarios of interest without requiring explicit scene modeling or rendering.

In the context of autonomous driving, this approach is especially attractive. Dashcam footage is widely available online, making it easier to collect realistic visual data compared to control signals or trajectory labels, such as is often the case in robotics. This abundance of video enables training powerful generative world models that can simulate diverse crash scenarios, unusual agent interactions, and complex environmental cues—all of which are difficult to script or collect in traditional pipelines. As we will explore in the Chapter 6, our work builds upon this emerging paradigm by designing a controllable video diffusion model for simulating crash scenarios from real-world dashcam videos. In the next section, we will dive deeper into the inner workings of diffusion models.

2.2 Video Diffusion Models

Diffusion models [12–14] have emerged as a powerful paradigm for generative modeling, particularly in the domain of image synthesis. They operate by learning to reverse a gradual noising process applied to data, generating high-fidelity samples through iterative denoising. Recent advances have extended these techniques to the video domain, where temporal consistency and spatial coherence are critical [7, 15, 16]. Diffusion models are not restricted to image synthesis; they can be applied to a wide range of data modalities, including video,

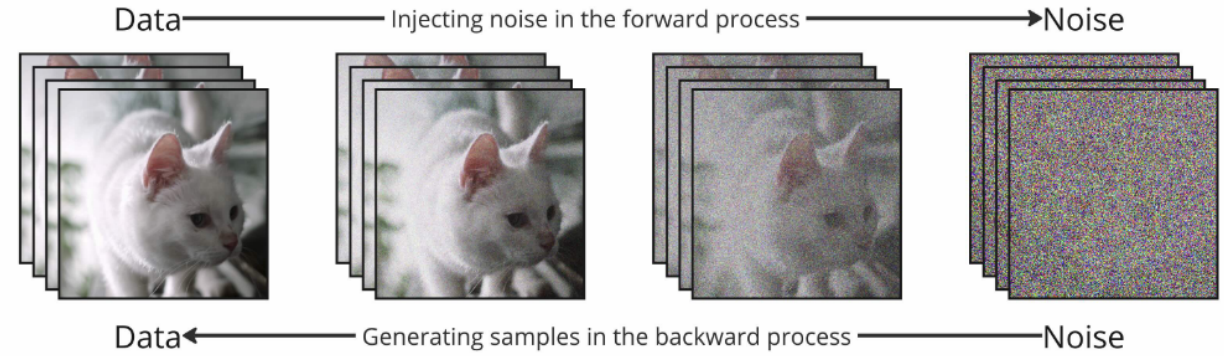


Figure 2.2 **The forward and reverse diffusion processes on video data**. A video is interpolated between clean data and pure noise. Figure source: [11]

audio, 3D shapes, and molecular structures. In the case of video, one can view the data as a three-dimensional tensor, where the two spatial dimensions (height and width) are extended by a third, temporal dimension. This temporal axis captures the evolution of visual content over time, enabling the model to learn both spatial structures and their dynamics. While the focus of this section is on video generation, the principles described extend naturally to other multi-dimensional data types.

At their core, denoising diffusion models consist of two processes: a forward diffusion process that gradually adds noise to input, and a reverse denoising process that learns to generate data by denoising, as shown in Figure 2.2. To generate a sample after training, we sample from a Gaussian distribution of random white noise and proceed to iteratively denoise this pure noise with a neural network that was specifically trained to guide noise toward a noise-free target distribution. Therefore the key element to the generative abilities of a diffusion model is the denoising process which is trained via an objective that learns to reverse the forward noising process. In this section, we will focus on two important diffusion model paradigms: denoising diffusion probabilistic models (DDPM) [14] and denoising diffusion implicit models (DDIM) [17].

Denoising Diffusion Probabilistic Model (DDPM). The forward diffusion process iteratively adds noise to an input video x_0 over T steps to gradually corrupt it towards pure Gaussian noise. This process is a Markov chain, where the degraded video x_t at time step t only depends on the video x_{t-1} from the immediate previous step. Formally, we model a video x_0 as a sample from the data distribution $q(x_0)$, and define a forward process that gradually corrupts it via Gaussian noise:

Algorithm 1 Training

repeat
  $x_0 \sim q(x_0)$
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta \left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t) \right\|^2$
until converged

Algorithm 2 Sampling

$x_T \sim \mathcal{N}(0, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  $z \sim \mathcal{N}(0, \mathbf{I})$ if $t > 1$, else $z = 0$
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z$
end for
return x_0

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}\, x_{t-1}, \beta_t \mathbf{I}), \tag{2.1}$$

where β_t is a small positive variance parameter controlling the random noise magnitude at each timestep t.

However, it is not really necessary to add Gaussian noise progressively, one step at a time. Instead, the distribution at an arbitrary time step t can be directly obtained from the initial input video x_0, which will speed up the sampling of corrupted samples:

$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t)\mathbf{I}), \tag{2.2}$$

where we define $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ and $\alpha_t = (1-\beta_t)$. The β_t values are carefully selected such that $\bar{\alpha}_T \to 0$ and $q(x_T) \approx \mathcal{N}(x_T; 0, \mathbf{I})$. From here, we can sample a corrupted data point x_t directly. Note that we can sample from a normal distribution like so: $x \sim \mathcal{N}(\mu, \sigma^2) \;\Rightarrow\; x = \mu + \sigma \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$, therefore:

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, \tag{2.3}$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ or random gaussian noise. We now have a distribution from which we can easily sample x_t for any time step $t \in [1, ..., T]$ given x_0, we represent this tractable distribution for the forward process as $q(x_{1:T} \mid x_0)$.

Now that we have defined the forward process, we can take a look at the reverse process. We consider the generative process $p_\theta(x_{0:T})$ defined by parameters θ. The process starts at time step T and works backwards to 0, where $p(x_T) = \mathcal{N}(x_T; 0, \mathbf{I})$. This prior is pure Gaussian noise and therefore does not depend on the parameters θ. The core part of the denoising process that must be learned is the transition kernel $p_\theta(x_{t-1} \mid x_t)$ which takes the form:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)), \tag{2.4}$$

where μ_θ is the mean parametrized by a neural network and variance Σ_θ is commonly fixed: $\Sigma_\theta(x_t, t) = \sigma^2\mathbf{I} = \beta_t\mathbf{I}$. Simply put, if we can learn $\mu_\theta(x_t, t)$, we can recover clean data by reversing the noising process.

Let's derive an objective function to learn this reverse process. Formally, the goal is to maximize the log-likelihood $\log p_\theta(x_0)$ of real data x_0 under our model parametrized by θ. However, to directly compute $\log p_\theta(x_0)$, it would require integrating over T latent variables $(x_1, ..., x_T)$, which is intractable:

$$\log p_\theta(x_0) = \log \int p_\theta(x_{0:T})\, dx_{1:T} \tag{2.5}$$

Therefore, we want to approximate the log-likelihood $\log p_\theta(x_0)$ via an optimization function. We can start by rewriting Equation 2.5 as follows:

$$\log p_\theta(x_0) = \log \int p_\theta(x_{0:T})\, dx_{1:T} \tag{2.6}$$

$$= \log \int p_\theta(x_{0:T}) \frac{q(x_{1:T} \mid x_0)}{q(x_{1:T} \mid x_0)}\, dx_{1:T} \tag{2.7}$$

$$= \log \mathbb{E}_q \left[\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right] \tag{2.8}$$

Using Jensen's inequality, we can rewrite this as an inequality by moving the log inside the expectation:

$$\log \mathbb{E}_q \left[\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right] \geq \mathbb{E}_q \left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right] \tag{2.9}$$

In variational inference, this is known as the Evidence Lower Bound (ELBO) $\mathcal{L}_{\text{ELBO}}$:

$$\log p_\theta(x_0) \geq \mathbb{E}_q \left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right] =: \mathcal{L}_{\text{ELBO}} \tag{2.10}$$

In the context of training our model, we can instead write this as a variational *upper* bound on the *negative* log-likelihood such that we express the objective as loss function $\mathcal{L}$ to minimize:

$$\mathcal{L} =: \mathbb{E}_q \left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right] \geq -\log p_\theta(x_0) \tag{2.11}$$

Finally we can expand the numerator and denominator which are respectively the reverse model:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t), \tag{2.12}$$

and the forward process:

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}), \tag{2.13}$$

So the loss function becomes:

$$\mathcal{L} = \mathbb{E}_q \left[-\log p(x_T) - \sum_{t=1}^{T} \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_t \mid x_{t-1})} \right] \geq -\log p_\theta(x_0) \tag{2.14}$$

Sohl-Dickstein et al. [13] show that this loss function can be rewritten as a sum of Kulback-Leibler divergences between the distributions of the forward and the backward steps:

$$\mathcal{L} = \mathbb{E}_q \left[\underbrace{D_{\mathrm{KL}}(q(x_T \mid x_0) \,\|\, p(x_T))}_{\mathcal{L}_T} + \sum_{t=1}^{T} \underbrace{D_{\mathrm{KL}}(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t))}_{\mathcal{L}_{t-1}} - \underbrace{\log p_\theta(x_0 \mid x_1)}_{\mathcal{L}_0} \right] \tag{2.15}$$

With this formulation, it is possible to calculate closed-form solutions for the Kulback-Leibler (KL) terms. The first loss term $\mathcal{L}_T$ is a constant and can therefore be ignored in the learning process: the conditional distribution $q(x_T \mid x_0)$ admits an analytical solution (Equation 2.2) and the prior $p(x_T)$ is fixed.

The second loss term $\mathcal{L}_{t-1}$, is the main loss term to optimize, and this formulation is useful as it contains the tractable posterior distribution $q(x_{t-1} \mid x_t, x_0)$ that can be expressed as a Gaussian by leveraging Bayes' rule:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\left(x_{t-1}; \tilde{\mu}_t(x_t \mid x_0), \tilde{\beta}_t \mathbf{I}\right), \tag{2.16}$$

where $\tilde{\mu}(x_t, x_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\, x_0 + \frac{\sqrt{\alpha_t}\,(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\, x_t \quad$ and $\quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\, \beta_t$.

Since both $q(x_{t-1} \mid x_t, x_0)$ (Equation 2.16) and $p_\theta(x_{t-1} \mid x_t)$ (Equation 2.4) are Normal distributions, the KL divergence has the following simple form:

$$\mathcal{L}_{t-1} = D_{\mathrm{KL}}\left(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t)\right) = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \left\| \tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t) \right\|^2 \right] + C \tag{2.17}$$

where C is a constant that does not depend on θ. Remember that $\tilde{\mu}_t(x_t, x_0)$ is the true mean of the posterior $q(x_{t-1} \mid x_0, x_t)$, and $\mu_\theta(x_t, t)$ is the learned mean of the reverse distribution $p_\theta(x_{t-1} \mid x_t)$. From this formulation, we can see that optimizing μ_θ to predict $\tilde{\mu}_t$ would

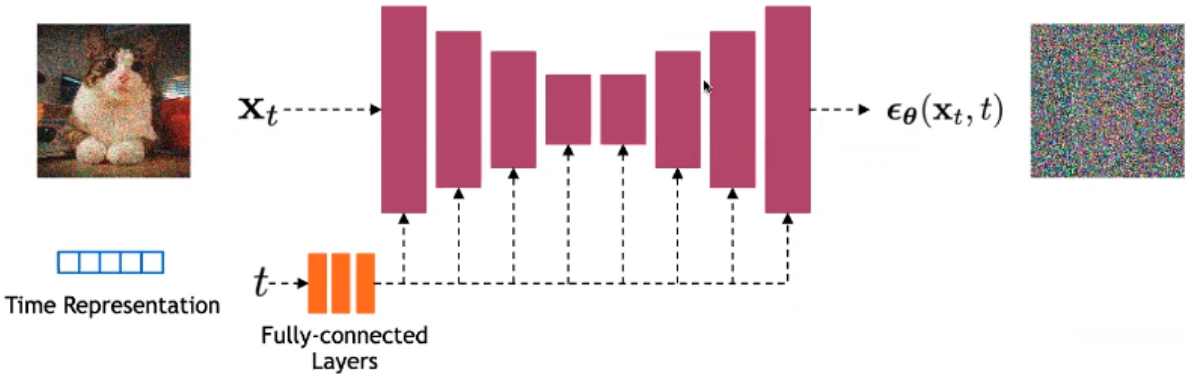


Figure 2.3 **Illustration of a U-Net-based denoiser $\epsilon_\theta(x_t, t)$ used in DDPMs.** The model takes a noisy input frame x_t and a timestep embedding t, and outputs an estimate of the noise ϵ used in the forward process. Figure source: `https://cvpr2022-tutorial-diffusion-models.github.io/`

minimize this loss function.

From here, Ho et al. [14], present the following relationship by reparameterizing the objective to predict ϵ:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \frac{1 - \bar{\alpha}_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) \tag{2.18}$$

In other words, instead of training the reverse process mean function approximator μ_θ to predict $\tilde{\mu}_t$, we modify its parametrization, so we can train to predict ϵ instead. This trick allows us to interpret the core of a DDPM model as a noise-predictor network $\epsilon_\theta(x_t, t)$ from which we can infer the mean $\mu_\theta(x_t, t)$.

Following this reparametrization, the DDPM authors propose to simplify the initial objective function from Equation 2.15 to a simple mean squared error (MSE) loss $\mathcal{L}_{\text{simple}}$:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{t}, \epsilon \sim \mathcal{N}(0,1)} \left[\|\epsilon - \epsilon_\theta(x_t, t))\|_2^2 \right], \tag{2.19}$$

where $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon$, $\epsilon \sim \mathcal{N}(0, 1)$, and $\bar{\alpha}_t$ at $t \in [1, T]$ controls the diffusion schedule. Here, ϵ_θ is the "denoiser", the core of the diffusion model, which predicts the noise added to a given input x_t at step t.

The training process detailed above is summarized in Algorithm 1, where we can observe the form of the simplified loss function from Equation 2.19. Sampling is summarized in

Algorithm 2 where samples are obtained via gaussian sampling using the mean μ_θ expressed in terms of ϵ_θ as shown in Equation 2.18 and adding random noise $\sigma_t z$, $z \sim \mathcal{N}(0, \mathbf{I})$ at each timestep.

The denoiser ϵ_θ is commonly implemented using a U-Net architecture, due to its strong inductive bias for multi-scale spatial data. However, recent work has also explored alternative backbones such as Vision Transformers. A typical U-Net-based denoiser used in diffusion models is illustrated in Figure 2.3, where the output is a prediction of the noise component ϵ added during the forward process.

Denoising Diffusion Implicit Models (DDIM) DDIM presents an alternative sampling procedure for diffusion models that significantly accelerates inference while maintaining compatibility with the standard DDPM training objective.

As discussed in the previous section, the reverse diffusion process in DDPM is modeled as a stochastic Markov chain where the model is trained to predict the noise component ϵ added to the clean image x_0 during the forward process. Sampling a new image involves drawing random noise at each reverse step, which requires hundreds or thousands of iterations to yield high-fidelity outputs.

DDIM preserves the same forward process as DDPM, including the training objective and learned noise prediction model $\epsilon_\theta(x_t, t)$, but introduces a deterministic, non-Markovian reverse process. The key idea lies in the fact that DDIM matches the marginal distribution $q(x_t|x_0)$ from DDPM, but allows for deterministic mappings between timesteps while denoising. This makes DDIM an implicit generative model: while it can sample from the data distribution, it does not define an explicit likelihood or use a variational bound, and thus does not require the stochasticity imposed by DDPM's ELBO.

DDIM generalizes the denoising process by constructing a family of deterministic and non-Markovian reverse processes that preserve the marginal distributions of DDPM at each timestep:

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\hat{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1}}\,\epsilon_\theta(x_t, t), \tag{2.20}$$

where $\hat{x}_0$ is estimated from x_{t_i} using the model's prediction $\epsilon_\theta(x_t, t)$:

$$\hat{x}_0 = \frac{x_t - \sqrt{1 - \bar{\alpha}_t}\,\epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}. \tag{2.21}$$

This formulation yields a deterministic trajectory from $x_T \sim \mathcal{N}(0, \mathbf{I})$ to x_0. We can observe

that this path is no longer Markovian as each step implicitly depends on the original estimate $\hat{x}_0$, rather than only x_t.

Though commonly used in a deterministic fashion, DDIM can be generalized to allow controllable stochasticity with the noise hyperparameter $\eta \in [0, 1]$, yielding the following generalized update:

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\hat{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1} - \eta^2\sigma^2}\,\epsilon_\theta(x_t, t) + \eta\sigma_t z, \tag{2.22}$$

where $z \sim \mathcal{N}(0, \mathbf{I})$, and the variance $\sigma_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$ is the same used in DDPM. At $\eta = 0$, the process is fully deterministic, and $\eta = 1$ recovers DDPM's stochastic sampling as shown in Algorithm 2. Tuning η allows trading off sample diversity for sample consistency and convergence speed.

A key advantage of DDIM's deterministic formulation is that it allows sampling from arbitrary subsequences of the diffusion schedule, effectively generating a video from less diffusion steps (i.e., less model calls). Instead of iterating over all T timesteps, DDIM defines a subsequence $\tau = (\tau_1, \tau_2, ..., \tau_S)$ where τ is an increasing sub-sequence of $[1, ..., T]$ of length $S \ll T$, where $\tau_1 \approx 1$ and $\tau_S \approx T$. This enables efficient generation in only 10-50 steps as opposed to something closer to 1000 steps in DDPM using the new update function:

$$x_{\tau_{i-1}} = \sqrt{\bar{\alpha}_{\tau_{i-1}}}\hat{x}_0 + \sqrt{1 - \bar{\alpha}_{\tau_{i-1}}}\,\epsilon_\theta(x_{\tau_i}, \tau_i), \tag{2.23}$$

In summary, DDIM provides a principled way to improve sampling efficiency and controllability in diffusion models while keeping the same training loss and learned model as DDPM, and therefore can be applied post hoc as a sampling method to any model trained with the standard DDPM loss. This compatibility makes it particularly attractive for applications requiring fast sampling, such as interactive generation or time-constrained inference. Moreover, the deterministic nature of DDIM sampling facilitates inversion — recovering the latent noise vector that generated a given sample — which is not straightforward in DDPM due to its stochastic transitions. This has led to its adoption in many downstream tasks, including image editing, interpolation, and conditional generation.

2.3 Controllable Generative Models

Recent progress in generative modeling has emphasized not only fidelity, but also controllability; the ability to guide outputs through structured input signals. In image generation, this includes control via text prompts, sketches, bounding boxes, semantic maps, and more. In diffusion models, classifier-free guidance (CFG) [18] and ControlNet [19] have been in-

troduced as effective methods to allow adaptable conditioning while preserving high-quality generation.

Classifier-Free Guidance Diffusion models can be guided to improve conditional generation quality by steering the generative process toward satisfying conditioning inputs (e.g., class labels, text prompts). A foundational approach, known as classifier guidance [20], does this by modifying the diffusion score estimate ϵ_θ using the gradient of a separately trained classifier. This method allows users to trade off between sample diversity and fidelity: increasing classifier influence can improve visual quality at the cost of reduced mode coverage. However, classifier guidance introduces practical limitations. Namely, it requires training and maintaining an auxiliary classifier, and this classifier must be differentiable and aligned with the generative task. That is to say, the classifier must be trained on noisy data so it is not possible to use an off-the-shelf pre-trained classifier. These requirements are often cumbersome and can constrain the flexibility of the model.

Classifier-free guidance offers a simpler and more flexible alternative that eliminates the need for an external classifier. CFG modifies the diffusion sampling process by linearly combining two score estimates: one from a conditional model and one from an unconditional model. Rather than training these models separately, CFG uses a single diffusion model trained in a joint conditional-unconditional fashion. This is achieved by randomly replacing the conditioning input with a null value (denoted as $c = \emptyset$) during training, causing the model to learn both behaviors simultaneously. This approach requires no additional networks and supports arbitrary conditioning modalities (e.g., text, segmentation maps, bounding boxes), as long as they can be masked out.

At inference time, denoising is guided by interpolating between the conditional noise estimate $\epsilon_\theta(\mathbf{x}_t, c)$ and the unconditional one $\epsilon_\theta(\mathbf{x}_t, \emptyset)$, scaled by a guidance strength parameter $\gamma \geq 1$. The resulting guided estimate is:

$$\hat{\epsilon}_\theta(\mathbf{x}_t, c) = \epsilon_\theta(\mathbf{x}_t, \emptyset) + \gamma \cdot (\epsilon_\theta(\mathbf{x}_t, c) - \epsilon_\theta(\mathbf{x}_t, \emptyset)) \,. \tag{2.24}$$

This formula can be interpreted as nudging the model in the direction that increases alignment with the conditioning input. By tuning γ, users can effectively balance between diversity (lower γ) and conditioning fidelity (higher γ), similar to temperature or truncation sampling in other generative models. CFG has become a standard technique in conditional diffusion models due to its simplicity, compatibility with a wide range of condition types, and ability to significantly boost sample quality with minimal overhead.

ControlNet While CFG enables flexible trade-offs between unconditional and conditional sampling during inference, it does not provide fine-grained spatial control over the generation process. To address this, ControlNet introduces a complementary architecture that allows explicit conditioning of the diffusion model on structured spatial inputs, such as edge maps, depth maps, pose keypoints, or segmentation masks.

The key idea in ControlNet is to augment a pretrained diffusion model (such as Stable Diffusion [21]) with a parallel, trainable network that mirrors the structure of the original U-Net backbone but is conditioned on an external control signal. During training, the original diffusion model is kept frozen, while the ControlNet learns to transform the control signal into additive feature modifications that are injected into the main U-Net at multiple intermediate layers. This allows the model to preserve the high-fidelity generative capabilities of the base model while introducing new controllable behaviors.

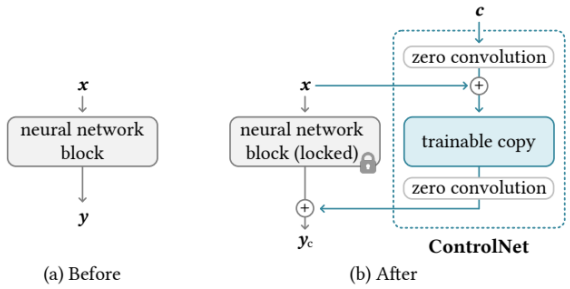


Figure 2.4 **ControlNet augments a frozen base model with a parallel trainable copy**. Figure source: [19].

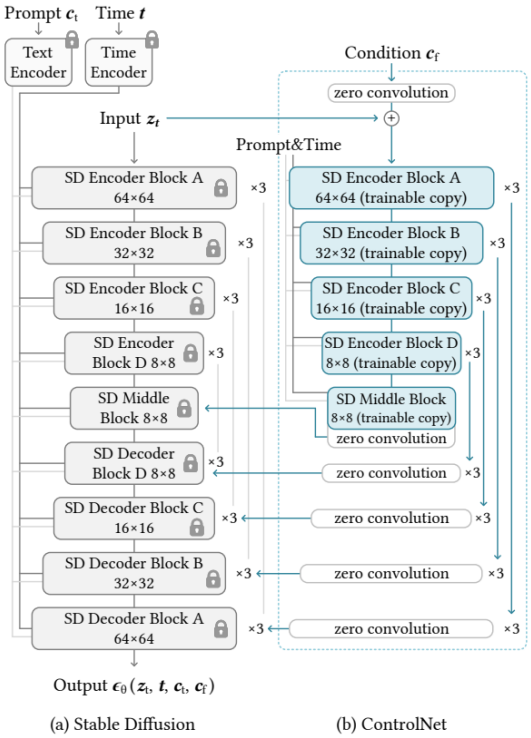


Figure 2.5 **Detailed architecture of ControlNet applied to Stable Diffusion**. Figure source: [19].

As illustrated in Figure 2.4, ControlNet augments a frozen base model by attaching a parallel, trainable network that mirrors the structure of the original U-Net. This trainable copy, known as the adapter, is initialized identically but learns independently, allowing it to process spatial conditioning signals such as segmentation maps, pose skeletons, or bounding boxes. The output of this parallel path influences the generation process by injecting learned features into the base model at various points, steering the denoising trajectory without altering the original model's parameters. This modular design makes ControlNet easily applicable to a wide range of diffusion-based backbones, requiring no changes to the pretrained model and offering flexible integration of structured inputs.

Figure 2.5 provides a more detailed view of ControlNet's architecture in the context of Stable Diffusion. Specifically, the ControlNet copies the encoder blocks and the middle block of the U-Net, while keeping the decoder blocks of the base model frozen and unchanged. The conditioning input $\mathbf{c}_\mathrm{f}$ (e.g., a segmentation mask) combined with the model input $\mathbf{z}_t$ (e.g., an encoded text prompt) is passed through the copied encoder and middle layers, producing intermediate feature maps at multiple resolutions. These are injected into the corresponding stages of the frozen base model using 1×1 convolutions which act as lightweight adapters. These layers are initialized to zero (i.e., "zero convolutions") so that the initial behavior of the model matches the original unconditioned diffusion model, and any learned influence of the conditioning signal is entirely data-driven. This helps stabilize training early on by staying close to the pre-trained model's behaviour. This careful design ensures that the generative prior remains intact while enabling effective and localized control over the generation process.

This plug-and-play approach to control has proven highly effective, enabling generation conditioned on spatial maps without retraining the full model. In our work, we adopt this architecture to steer generation toward safety-critical events using conditioning signals such as bounding box trajectories and crash type labels.

2.4 Video Evaluation Metrics

Evaluating the quality of generated videos requires metrics that can capture both the overall realism of the sequence and the fidelity of individual frames. In this thesis, we employ two complementary categories of metrics. Distributional video-level metrics, such as the Fréchet Video Distance (FVD) [22] and the JEPA Embedding Distance (JEDi) [23], assess the realism of generated videos by comparing their distribution to that of real videos in a learned feature space. These metrics are sensitive to both spatial content and temporal consistency, making them well-suited for capturing motion quality and scene coherence. In parallel, we use frame-level metrics, including Learned Perceptual Image Patch Similarity (LPIPS) [24], Structural Similarity Index (SSIM) [25], and Peak Signal-to-Noise Ratio (PSNR), which quantify visual fidelity by comparing each generated frame to its corresponding ground truth. These are averaged over all frames to produce a single score per video. Together, these metrics provide a comprehensive evaluation: distributional metrics capture perceptual realism and temporal dynamics, while frame-level metrics measure precise visual similarity. The following subsections detail the formulation and interpretation of each metric.

2.4.1 Video metrics: FVD and JEDi

FVD and JEDi are both video-level metrics that work in similar ways, yet have key distinctions between them. Both metrics measure the distance between the distribution of generated videos P with the distribution of real videos Q (where lower is better). In the case of FVD, this is done by embedding the two distributions in the feature space of a pre-trained Inflated 3D ConvNet (I3D) [26] and computing the Fréchet distance (FD) under the assumption of a multivariate Gaussian. The Fréchet distance, also known as 2-Wasserstein distance, measures how similar a distribution P is from a distribution Q and is defined as the minimum distance between all pairs of random variables x and y from the distributions. For FVD, we assume P and Q as multivariate Gaussians:

$$d_{Fréchet}(P,Q) = |\mu_P - \mu_Q|^2 + \mathrm{Tr}\left(\Sigma_P + \Sigma_Q - 2(\Sigma_P\Sigma_Q)^{1/2}\right) \tag{2.25}$$

where μ_P and μ_Q are the means, and Σ_P and Σ_Q are the covariance matrices of the two Gaussian distributions. "Tr" refers to the trace operation of a square matrix, which is the sum of its diagonal elements.

JEDi, on the other hand, addresses limitations of FVD by relaxing the Gaussian assumption, employing features from a Joint Embedding Predictive Architecture (JEPA) [27], and using Maximum Mean Discrepancy (MMD) [28] with a polynomial kernel to improve both temporal sensitivity and sample efficiency. Specifically, it embeds the two distributions (P and Q) into the JEPA feature space and uses MMD to compute the distance between the two sets of features. The advantage of MMD as opposed to Fréchet distance as a distance metric is that it is distribution-free, and thus requires no assumptions about the underlying distributuons of P and Q. MMD is a general class of kernel-based sample tests that maximize the mean difference between samples from two distributions by optimizing over all data transformations f within a function space $\mathcal{F}$. Formally, Given two sets of features, $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ and $Y = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$, sampled from P and Q, $d^2_{MMD}(P,Q)$ with a given kernel, k, is given by [28]:

$$\hat{d}^2_{MMD}(X,Y) = \frac{1}{m(m-1)}\sum_{i=1}^{m}\sum_{\substack{j=1\\j\neq i}}^{m} k(\mathbf{x}_i,\mathbf{x}_j) + \frac{1}{n(n-1)}\sum_{i=1}^{n}\sum_{\substack{j=1\\j\neq i}}^{n} k(\mathbf{y}_i,\mathbf{y}_j) - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} k(\mathbf{x}_i,\mathbf{y}_j). \tag{2.26}$$

Using MMD along with the JEPA feature space allows JEDi to converge to a reliable value with much fewer samples compared to FVD (I3D + FD) [23].

2.4.2 LPIPS, SSIM, and PSNR for Video Evaluation

In addition to video-level measures, frame-level metrics are commonly used to assess perceptual quality with respect to ground truth frames. Examples include LPIPS, SSIM, and PSNR. These metrics capture different aspects of visual correspondence—ranging from perceptual similarity to structural consistency and pixel-level fidelity—providing finer-grained insight into how well generated frames preserve appearance and local coherence. The following sections describe each of these metrics in detail.

Learned Perceptual Image Patch Similarity (LPIPS) The LPIPS metric is designed to evaluate the perceptual similarity between two images by comparing the responses of a deep neural network to each image. Unlike pixel-wise metrics, LPIPS leverages learned features from intermediate layers of a pretrained network (such as VGG [29]), making it more aligned with human visual perception.

Formally, given two images x and y, the LPIPS distance is defined as:

$$\text{LPIPS}(x, y) = \sum_l \frac{1}{H_l W_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} \|w_l \odot (\phi_l(x)_{hw} - \phi_l(y)_{hw})\|_2^2 \tag{2.27}$$

Here, $\phi_l(\cdot)$ denotes the activation at layer l of the pretrained network, and $\phi_l(x)_{hw} \in \mathbb{R}^C$ is the feature vector at spatial location (h, w) with C channels. The scalars H_l and W_l are the height and width of the feature map at layer l, and $w_l \in \mathbb{R}^C$ is a learned channel-wise weight vector applied elementwise via the Hadamard product $\odot$. The result is an average over spatial and channel dimensions of the squared ℓ_2 norm between features from the two images.

To extend LPIPS to video, where a generated video $\{x_t\}_{t=1}^T$ and a ground-truth video $\{y_t\}_{t=1}^T$ consist of T frames, the metric is averaged over all frames as:

$$\text{LPIPS}_{\text{video}} = \frac{1}{T} \sum_{t=1}^{T} \text{LPIPS}(x_t, y_t) \tag{2.28}$$

LPIPS measures perceptual distance; smaller values indicate higher visual similarity.

Structural Similarity Index Measure (SSIM) SSIM is a traditional image similarity metric that assesses the structural similarity between two images by comparing luminance, contrast, and structural information. It is based on the hypothesis that the human visual system is highly adapted to extract structural information from visual scenes. Given two

grayscale images x and y, SSIM is defined as:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.29}$$

In this equation, μ_x and μ_y are the mean intensities of images x and y, respectively. The terms σ_x^2 and σ_y^2 denote the variances of x and y, while σ_{xy} is their covariance. Constants C_1 and C_2 are used to stabilize the division when the denominators are small and are typically set relative to the dynamic range of the image (e.g., $C_1 = (0.01L)^2$, $C_2 = (0.03L)^2$ for pixel values in range $[0, L]$).

To compute SSIM over a video, the image-level SSIM is computed for each corresponding frame and averaged temporally (similar to LPIPS, shown in Equation 2.28). SSIM values lie in the range $[0, 1]$, with higher values indicating greater structural similarity between the generated and ground-truth videos.

Peak Signal-to-Noise Ratio (PSNR) PSNR is a classical metric used to measure the pixel-level fidelity between two images, based on the mean squared error (MSE). Although it is easy to compute and widely used, it does not always correlate well with human perceptual quality, especially for complex textures or structural misalignments. Given two images x and y of size $H \times W$, the PSNR is defined as:

$$\text{PSNR}(x, y) = 10 \cdot \log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}(x, y)}\right) \tag{2.30}$$

where MAX is the maximum possible pixel value (e.g., 255 for 8-bit images), and $\text{MSE}(x, y)$ is the mean squared error between the two images:

$$\text{MSE}(x, y) = \frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}(x_{ij} - y_{ij})^2 \tag{2.31}$$

To extend PSNR to videos, it is standard to compute the PSNR for each frame and then average the scores over time. Higher PSNR values indicate better pixel-level similarity. However, high PSNR does not guarantee good perceptual quality and may not penalize temporal artifacts or structural inconsistencies effectively, only pixel-level fidelity to ground-truth.

CHAPTER 3 BEHAVIOUR MODELING FOR SAFETY-CTICAL AV SCENARIOS

This chapter draws upon two prior publications, CtRL-Sim [30] and Scenario Dreamer [31], to which the author contributed as a co-author. Although the work was not led by the author, this chapter functions both as a targeted literature review and as a conceptual bridge to the core contributions presented in subsequent chapters.

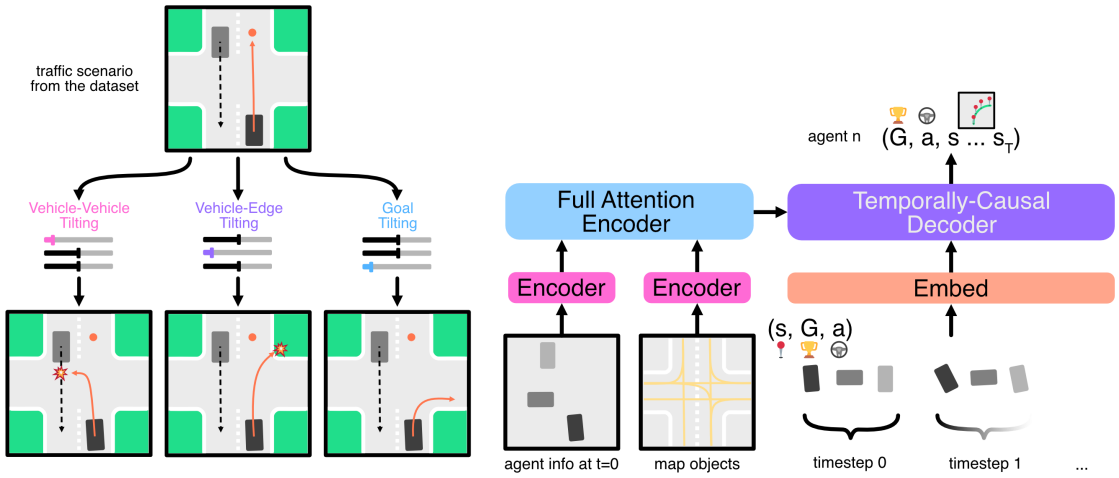
3.1 Introduction

AVs must operate safely and reliably in a wide range of situations, including rare and dangerous scenarios such as near-miss events, collisions, and unexpected pedestrian interactions. Due to the difficulty and ethical concerns associated with collecting real-world data for such events, simulation has emerged as a critical tool for the development and validation of AV systems. A key requirement for effective simulation is the ability to model the behaviour of other agents (e.g., vehicles, pedestrians, cyclists) in a way that is both realistic and reactive to the AV's own behaviour.

A common abstraction used in AV simulation is the bird's-eye view (BEV) representation, in which the environment is projected onto a 2D top-down plane. This representation is favored for its interpretability, efficiency, and alignment with modern AV perception and planning pipelines. Within this context, a core challenge is to generate plausible agent trajectories and interactions in a controllable and efficient manner. We represent agents (vehicles, pedestrians, cyclists, etc.) as individual entities described by there 2D size, position, orientation, velocity and acceleration. This chapter explores how generative AI, specifically diffusion models, autoregressive models, and models based on offline reinforcement learning, can be used to simulate controllable behaviour and generate safety-critical driving scenarios in a data-driven manner.

While traditional driving simulators often replay logged trajectories or use rule-based policies for background traffic agents, these methods fall short in critical ways. Replayed agents are not reactive to AV behaviour, leading to unrealistic interactions. Rule-based agents offer reactivity, but lack realism and diversity. To overcome these limitations, we explore generative AI methods for simulating controllable and reactive learnt agent behaviour in BEV, providing a foundation for constructing rich and customizable safety-critical scenarios.

3.2 Controllable Behaviour Simulation



(a) **Usage** (b) **Model Overview**

Figure 3.1 **CtRL-Sim usage and model overview. (left)** CtRL-Sim allows for controllable agent behaviour from existing datasets (e.g., Waymo Open Motion Dataset [32]). This allows users to create interesting edge cases for testing and evaluating AV planners. **(right)** The agent and map data at $t = 0$ are encoded and fed through a Transformer encoder as context for the decoder, similar to [33]. Trajectories are arranged first by agents, then by timesteps, embedded, and fed through the decoder. For each agent, we encode (s_t, G_t, a_t) (i.e. state, return-to-go, action) and we predict from these $(G_t, a_t, s_{t+1}, \ldots, s_T)$. Figure source: [30].

To address the lack of reactivity and controllability in traditional simulators, CtRL-Sim [30] introduces a novel framework for simulating driving behaviours using return-conditioned offline reinforcement learning. Rather than replaying actions from pre-recorded logs, CtRL-Sim learns a return-conditioned policy from real-world data, enabling agents to respond to changes in the AV's actions and exhibit varied behaviour at test time.

In CtRL-Sim, the driving behaviour of an agent is governed by a policy $\pi(a_t \mid s_t, m, s_g)$, which predicts an action a_t given the current state s_t, the map context m, and a goal state s_g. The evolution of the scene is modeled using a forward transition function $\mathcal{P}(s_{t+1} \mid s_t, a_t)$, which describes how the agent's state changes in response to actions. In practice, this transition model is realized via the physics-based Nocturne simulator [34], which ensures that the dynamics of agents are physically plausible.

The core learning problem is to model the driving policy π such that it can both reproduce realistic driving behaviours and allow for control over agent intent, especially to simulate rare or counterfactual events like crashes. This is formulated within the offline reinforcement

learning (offline RL) framework. In this setting, the agent does not interact with the environment during training. Instead it learns from a fixed dataset $\mathcal{D}$ of past driving trajectories. Each trajectory τ_i in the dataset consists of sequences of states, actions and rewards:

$$\tau_i = \{(s_0, a_0, r_0), (s_1, a_1, r_1), \ldots, (s_T, a_T, r_T)\} \tag{3.1}$$

where $s_t \in \mathcal{S}$ are states, $a_t \in \mathcal{A}$ are actions, and $r_t \in \mathbb{R}$ are scalar rewards. These trajectories are assumed to have been generated by a behaviour policy $\pi_B(a_t \mid s_t)$, which may be suboptimal.

The agent's objective is to learn the new policy $\pi(a_t \mid s_t)$ that achieves higher cumulative reward (called the return-to-go) than the trajectories in the dataset. The return-to-go from timestep t is defined as:

$$G_t = \sum_{t'=t}^{T} r_{t'} \tag{3.2}$$

In summary, the goal of the offline RL framework in CtRL-Sim is to learn a high-performing policy that generalizes beyond the behaviour observed in the offline dataset $\mathcal{D}$, while also enabling controllable deviations from typical driving patterns to synthesize long-tail and safety-critical scenarios. Note that in the approach detailed above, we describe the behaviour of a single agent for simplicity, but CtRL-Sim extends this to the multi-agent setting by jointly modeling all states, actions and rewards, effectively processing multiple agents in parallel.

At the core of CtRL-Sim is an encoder-decoder Transformer model [35]—drawing inspiration from prior work that use Transformers for sequence modeling via RL [36, 37]—trained on a dataset of physically plausible trajectories simulated using a physics-enhanced version of the

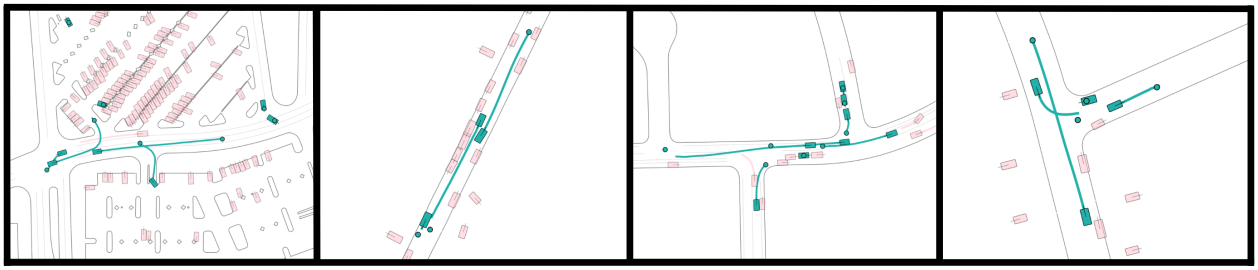


Figure 3.2 **Qualitative results of multi-agent simulation with CtRL-Sim.** The teal agents are controlled by CtRL-Sim, and other agents in pink are set to log-replay through physics. Figure source: [30].

Nocturne simulator. Each trajectory is annotated with multiple reward signals: goal-reaching success, collisions, and lane boundary violations. These diverse supervision channels enable the model to capture agent behaviour across several safety-relevant dimensions.

The model is trained to predict not only the next action but also the entire future state sequence and return-to-go, which serves as a form of auxiliary regularization [38]. Formally, the model is trained to estimate the following joint distribution:

$$p_\theta(s_{t+1:T}, a_t, G_t \mid s_t, s_g) = p_\theta(s_{t+1:T} \mid s_t, s_g, G_t, a_t) \cdot \pi_\theta(a_t \mid s_t, s_g, m, G_t) \cdot p_\theta(G_t \mid s_t, s_g), \quad (3.3)$$

where the model first samples a return-to-go G_t conditioned on the current state s_t and goal s_g:

$$G_t \sim p_\theta(G_t \mid s_t, s_g), \quad (3.4)$$

then samples an action a_t conditioned on the state, goal, return G_t, and map information m:

$$a_t \sim \pi_\theta(a_t \mid s_t, s_g, m, G_t). \quad (3.5)$$

Finally the model predicts the full sequence of future states of agents $s_{t+1:T}$:

$$s_{t+1:T} \sim p_\theta(s_{t+1:T} \mid s_t, s_g, G_t, a_t). \quad (3.6)$$

By predicting the next action for an agent, we can control its motion through time, all while being reactive to other agents. Figure 3.2 shows example rollouts of agents controlled by CtRL-Sim.

Note in Equation 3.5, the sampled action comes from a modified expression of the policy π_θ which is conditioned on the return G_t. This return-conditioned policy formulation is a key component of the CtRL-Sim framework as it allows the explicit alignment between actions and their corresponding returns. Indeed, this sampling process reflects an imitative policy: the model learns to act in accordance with the distribution of returns observed in the training data. To enable user-driven control over agent behaviour at test time, CtRL-Sim allows for exponentially tilting the return distribution [38–40], effectively biasing the agent toward higher or lower reward behaviours. Specifically, return values are sampled from a modified distribution:

$$G'_t \sim p_\theta(G_t \mid s_t, s_g) \cdot \exp(\kappa G_t), \quad (3.7)$$

where κ is an inverse temperature parameter. Positive values of κ bias the agent toward high-

return (e.g., cautious) behaviour, while negative values steer the agent toward low-return (e.g., risky or adversarial) behaviour. This mechanism provides a direct and interpretable means of controlling behaviour generation across a spectrum of outcomes.

This controllability is critical for safety-focused simulation. For example, by negatively tilting the collision reward, one can simulate near-miss or actual crash scenarios in a principled and data-driven manner. The result is a flexible simulator that can generate realistic and dangerous scenarios to test the robustness of AV planning systems under edge cases. We show the results of our controllability evaluation in Figure 3.3. For each reward dimension c, we exponentially tilt κ_c between -25 and 25 and observe how this affects the corresponding metric of interest. Collision rate, offroad rate and goal success rate are computed over 1000 scenes among which we measure the rate at which the agent of interest is involved in a collision with another agent, crosses a outside road boundary, and reaches its goal destination, respectively.

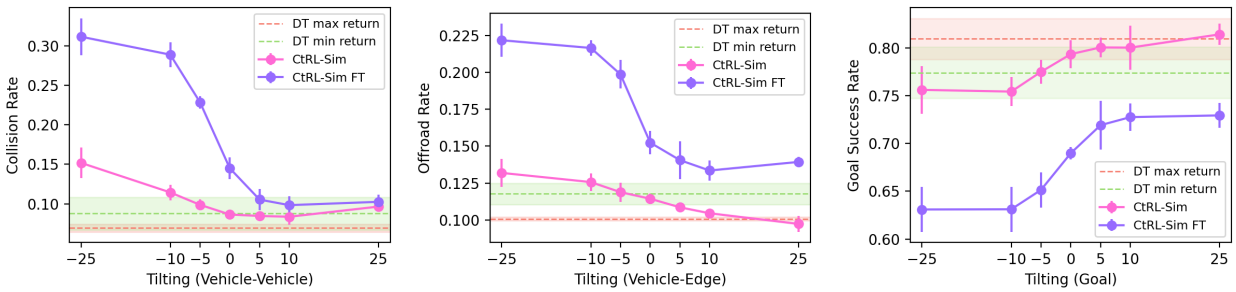


Figure 3.3 **Effects of exponential tilting.** Comparison of CtRL-Sim base model (magenta) and fine-tuned model (purple) across different reward dimensions. Rewards range from -25 to 25 for vehicle-vehicle collision (**left**), vehicle-edge collision (**middle**), and goal reaching (**right**). Results show smooth controllability, with fine-tuning enhancing this effect. We report mean ± std over 5 seeds. Figure source: [30].

3.3 Generating Realistic Environments

While CtRL-Sim allows for controllable behaviour generation, it still relies on ground-truth map data and initial scene configurations. To remove this dependency and enable fully generative simulation, we introduced Scenario Dreamer [31]. This system complements CtRL-Sim by generating the initial traffic scene itself—including lane graph topology and agent bounding boxes—using a vectorized latent diffusion model.

Unlike previous approaches that rasterize the scene (which is inefficient and lossy), Scenario Dreamer processes vectorized scene elements directly. It uses a Transformer-based diffusion

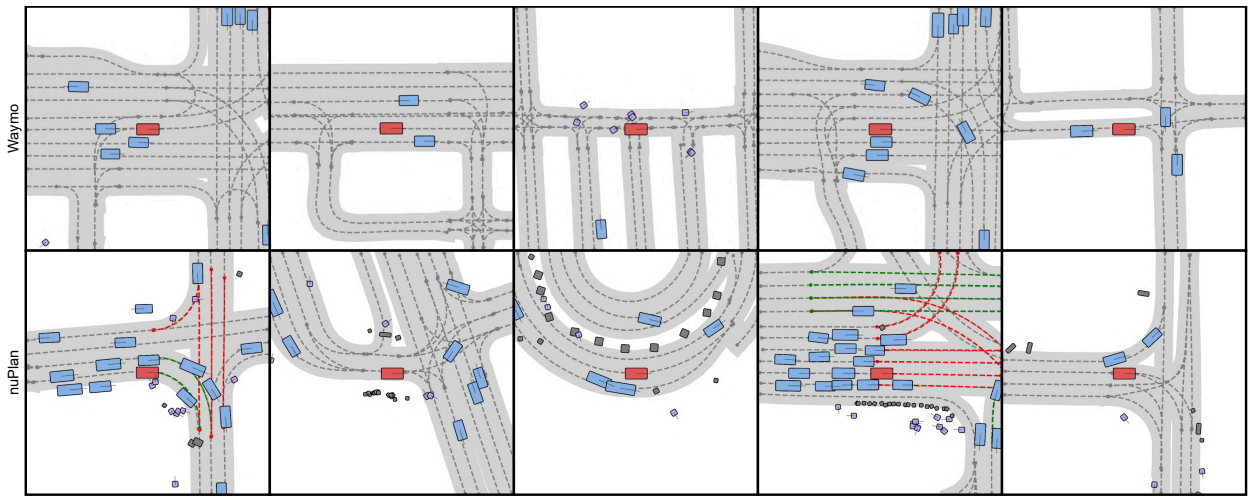


Figure 3.4 **Vectorized environments generated by Scenario Dreamer** with the proposed vectorized latent diffusion model trained on the Waymo dataset (top row) and nuPlan dataset (bottom row). Vehicles are drawn in blue and red (for the ego agent), while pedestriand are drawn in purple and static objects (e.g., traffic cones, barricades, telephone poles, etc.) are drawn as grey squares. For the model trained on the nuPlan dataset, the traffic light state of lanes is also depicted in red and green on the lanes. Figure source: [31].

model trained on datasets like Waymo and nuPlan to produce high-fidelity environments. It also supports scene extrapolation via inpainting, enabling generation of arbitrarily long driving environments by stitching together scene tiles.

Scenario Dreamer begins by generating the initial driving scene in a BEV coordinate system. This scene includes both the initial state of traffic participants and the structure of the road layout, all within a fixed spatial region. Specifically, the model operates over a 64m×64m field of view, denoted as F, which is centered and rotated to align with the orientation of the ego vehicle. The region F is further decomposed into two subregions: a forward region F_P, which spans 32m×64m ahead of the ego, and a rear region F_N of the same size but located behind.

The goal of the initial scene generator is to sample a complete scene from the distribution $p(\mathcal{I}_F)$, where a scene $\mathcal{I}_F$ is composed of two components: a set of objects $\mathcal{O}$ and a map structure $\mathcal{M}$, both contained within the spatial region. The object set $\mathcal{O} = \{\mathbf{o}_i\}_{i=1}^{N_o}$ includes all relevant actors and static entities such as vehicles, pedestrians, cyclists, and traffic cones. Each object $\mathbf{o}_i$ is encoded as an 8-dimensional vector representing the agent's 2D position, speed, orientation (via sine and cosine of the heading), physical dimensions (length and width), and semantic class.

The map component $\mathcal{M} = \{\mathcal{L}, \mathbf{A}\}$ captures road topology. Here, $\mathcal{L} = \{\mathbf{l}_i\}_{i=1}^{N_l}$ is a set of

lane centerlines, with each $\mathbf{l}_i$ represented as a fixed-length sequence of 20 two-dimensional coordinates. The connectivity between lanes is expressed via a third-order binary tensor $\mathbf{A} \in \{0,1\}^{N_l \times N_l \times 4}$ that encodes successor, predecessor, left-neighbor, and right-neighbor relations across lanes.

The model can be conditioned to generate a desired number of agents and lanes for diverse driving scenarios. Figure 3.4 shows several examples of generated scenes exhibiting diverse lane structures and plausible road user positioning.

To enable scalable scene generation across larger areas, the model supports sampling from a conditional distribution $p(\mathcal{I}_{F_P} \mid \mathcal{I}_{F_N})$. This allows for forward expansion of the generated environment by conditioning on an existing region, making it possible to stitch together multiple spatial windows to simulate extended driving scenarios.

For behaviour simulation, Scenario Dreamer integrates the CtRL-Sim model to control agents after the initial scene is generated. This results in a fully generative, closed-loop simulator where both the environment and agent trajectories are synthesized from scratch, yet remain realistic and controllable.

Scenario Dreamer addresses the challenge of realistic AV scenario generation by learning to synthesize diverse driving environments in a data-driven manner. By modeling both the spatial layout of the road network and the distribution of dynamic agents, it enables the creation of plausible initial scenes without relying on manually designed maps or scripted object placements. This complements the broader goals of this thesis by expanding the scope of generative simulation beyond agent behaviour to include the foundational structure of the scene itself, thereby enabling more complete and realistic simulation pipelines for safety-critical autonomous vehicle testing.

3.4 Conclusion

The works presented in this chapter—CtRL-Sim and Scenario Dreamer—lay the foundation for a data-driven simulation framework that supports the scalable and controllable generation of safety-critical driving scenarios in BEV. CtRL-Sim enables fine-grained behaviour control of traffic participants through return-conditioned offline reinforcement learning, allowing for the targeted creation of both nominal and adversarial driving behaviours. Scenario Dreamer extends this capability by generating the initial simulation environment itself, thereby removing the dependence on fixed, pre-recorded driving logs or annotated HD maps. Together, these two components provide a flexible generative pipeline for creating long-tail driving scenes from scratch, directly in the abstract BEV representation space.

However, while BEV-based simulation offers interpretability and structured control, it abstracts away important aspects of real-world perception, such as visual complexity, occlusions, and sensor noise. To support end-to-end autonomous vehicle systems, especially those relying on raw sensory input, we argue that generating driver-perspective videos of safety-critical scenarios, such as crashes, is a crucial next step. The remainder of this thesis builds on the foundations established by CtRL-Sim and Scenario Dreamer, and investigates how generative models can be extended to produce photorealistic, controllable driving videos, enabling simulation at the pixel level to support perception-driven AV stacks.

CHAPTER 4 CTRL-V: CONTROLLABLE AV VIDEO GENERATION

This chapter draws from Ctrl-V [41], a prior work to which the author contributed as a co-author. While the work was not led by the author, it is presented here in detail as this work is directly extended and adapted in the core contribution of this thesis in the following chapter.

4.1 Introduction

Simulation has long been a cornerstone of AV development, particularly for tasks such as planning and control. Traditionally, these simulations operate in an abstract BEV representation, where the positions and intentions of agents can be easily visualized, manipulated, and measured. The BEV formulation offers strong structural priors and facilitates efficient behavior modeling, as explored in Chapter 3 through works such as CtRL-Sim and Scenario Dreamer. However, while BEV simulation is useful for generating and analyzing agent behaviors, it abstracts away the rich visual content that is essential to perception-based AV systems.

Modern AV stacks are increasingly end-to-end [42], operating directly on raw sensor inputs, like images or video streams, to perform tasks such as driving planning and control. For these systems, the ability to generate realistic and controllable visual inputs is extremely useful for training and testing. This becomes especially important in long-tail or safety-critical situations, such as collisions or near-misses, where real-world data is scarce due to ethical, legal, and logistical constraints.

In this context, video generation offers a promising direction. By synthesizing realistic driving scenes directly at the pixel level, we can enable new forms of training, testing, and counterfactual analysis for AV perception systems. However, a key challenge lies in controllability—the ability to specify what happens in the video, such as where objects move or how they interact with one another. Without such control, generated videos lack the specificity needed for meaningful AV evaluation.

This chapter explores how controllable video generation can be achieved with bounding box guidance, focusing our recent model called Ctrl-V. Ctrl-V establishes a powerful and flexible framework for generating videos where dynamic agents follow user-specified trajectories. This represents a critical step toward the ultimate goal of this thesis: generating realistic crash videos under fine-grained control.

4.2 Overview of Controllable Video Generation with Ctrl-V

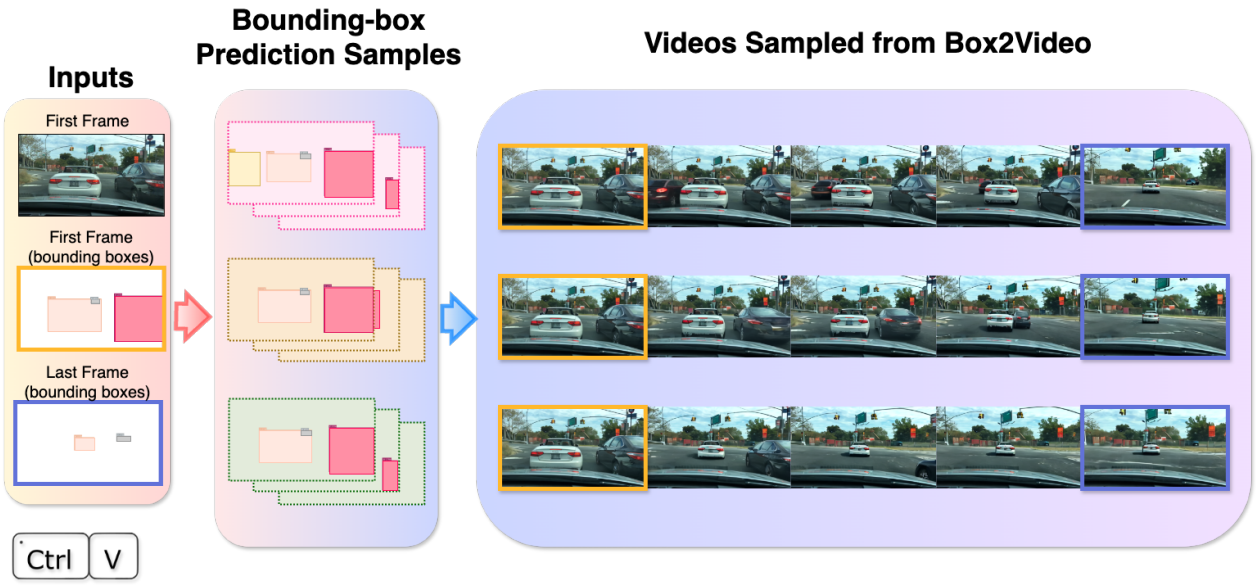


Figure 4.1 **Overview of Ctrl-V's generation pipeline. (Left)** Inputs: Our inputs include an initial frame, its corresponding bounding-box image and the final frame's bounding-box image. **(Middle)** Bounding-box generation samples: We illustrate three different sequences generated from our diffusion based bounding-box motion generation model. **(Right)** Videos sampled from our Box2Video diffusion model: Our Box2Video model conditions on the generated bounding-box videos to produce the final video clips. Figure source: [41].

Ctrl-V [41] is a generative model designed to synthesize videos where object motion is explicitly controlled via user-specified bounding box trajectories. The model is particularly suited to applications like autonomous driving, where dynamic scenes with multiple interacting agents are common, and where safety-critical scenarios need to be simulated with high fidelity. Ctrl-V is built around a modular two-stage architecture (as illustrated in Figure 4.1):

1. **Bbox Generator:** Given a static input image and target bounding box frames at the beginning and end of the sequence, this module generates a temporally coherent sequence of bounding boxes that represent plausible motion trajectories for one or more objects in the scene. This component is especially useful when no ground-truth trajectory annotations are available and supports generative simulation from static scenes.

2. **Box2Video Model**: This is the core generative module, responsible for synthesizing full-frame videos where objects move according to a specified bounding box trajectory.

> Given an initial reference frame and a bounding box sequence, the model generates a temporally consistent video that aligns the visual motion of objects with the spatial and temporal constraints imposed by the bounding box frame sequence.

The overall pipeline is flexible: users may either provide their own trajectories or use the Bounding Box Generator to synthesize new ones. The final video is produced by the Box2Video model, which learns to interpret these control signals and produce visually realistic outputs that adhere to the intended object dynamics.

In the context of autonomous driving, Ctrl-V enables the creation of fully synthetic video scenarios where specific object interactions (such as a vehicle turning into another's lane or a pedestrian entering the street) can be scripted and rendered with high spatial and temporal fidelity. This controllability makes Ctrl-V a compelling candidate for developing AV benchmarks, training perception systems, and performing counterfactual reasoning where pixel-level and split-second precision can make a great difference.

In the following sections, we dive into each component of the model in greater detail, we will mainly focus on the Box2Video model which is the core element of Ctrl-V and also the model that forms the basis for the Ctrl-Crash framework that will be later introduced in Chapter 6. We will begin with Box2Video, then follow it up with the Bbox Generator which offers a solution to generating the conditioning necessary for the video generation process.

4.3 Box2Video Model: Controllable Video Synthesis from Bounding Boxes

The Box2Video model is the central generative module in Ctrl-V, enabling the transformation of structured object motion, expressed as rendered bounding box sequences, into realistic video content. This model is responsible for producing dynamic, visually coherent scenes in which objects follow user-specified trajectories. In the context of AV simulation, this provides a crucial capability: it allows users to generate synthetic driving videos with explicit and interpretable control over object behaviors, bridging the gap between low-level perception and high-level scenario design.

4.3.1 Problem Formulation

The task tackled by the Box2Video model is to generate a temporally consistent video conditioned on two inputs: an initial reference frame $\boldsymbol{f}^{(0)}$, which captures the visual appearance and layout of the scene at the start of the sequence, and a rendered video of bounding box frames $\boldsymbol{f}_{bbox}$ that encodes object motion over time. These control sequences convey not only

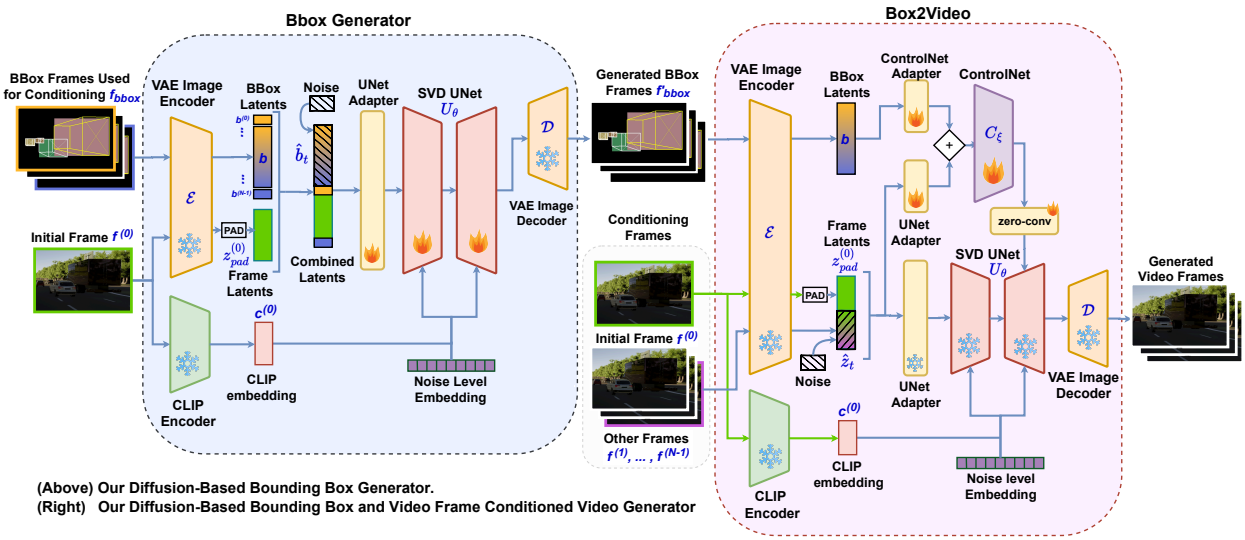


Figure 4.2 **Ctrl-V architecture diagram.** The diagram illustrates two components of **Ctrl-V**: (left) the Bbox Generator and (right) Box2Video. For both models, we use a frozen, off-the-shelf VAE to encode images into latent space ($\mathcal{E}$) and decode them back into pixel space ($\mathcal{D}$). During training, (1) the Bbox Generator (Sec. 4.4) learns to denoise the noisy bounding box frame latents $\hat{\boldsymbol{b}}_t$, conditioned on the first $\boldsymbol{b}^{(0)}$ and last $\boldsymbol{b}^{(N-1)}$ bounding box frame latents and the padded initial frame latent $\boldsymbol{z}_{pad}^{(0)}$ and (2) the Box2Video (Sec. 4.3) denoises the target frame latents $\hat{\boldsymbol{z}}_t$ by conditioning on the initial frame's latent $\boldsymbol{z}_{pad}^{(0)}$ (input to the SVD UNet) and the bounding box frame latents $\boldsymbol{b}$ (input to the ControlNet). Figure source: [41].

spatial information about where objects are expected to be, but also semantic cues such as object class, identity, and orientation. The goal of the model is to synthesize a coherent video that respects these motion constraints while preserving scene realism and dynamic consistency. The resulting video should depict each object moving in alignment with its specified trajectory, embedded naturally in the environment, and interacting appropriately with surrounding context.

4.3.2 Model Architecture

The Box2Video architecture, illustrated in the right block of Figure 4.2, builds on the Stable Video Diffusion (SVD) framework [7], an image-to-video latent diffusion model designed for high-quality temporal synthesis. In SVD, a single input image $\boldsymbol{f}^{(0)}$ serves as the initial condition, and the model generates a sequence (i.e., video) $\boldsymbol{f}_{out} = [\boldsymbol{f}^{(0)}, \ldots, \boldsymbol{f}^{(N)}]$, where N is the number of predicted future frames.

SVD operates in a latent space, meaning that the denoising and diffusion processes act on

latent representations $\boldsymbol{z}$ rather than directly on pixel space. A variational autoencoder (VAE) encoder $\mathcal{E}$ maps each video frame $\boldsymbol{f}^{(i)}$ into a latent $\boldsymbol{z}^{(i)}$, and a VAE decoder counterpart $\mathcal{D}$ reconstructs it back to pixel space:

$$\mathcal{D}\Big(\mathcal{E}(\boldsymbol{f}^{(i)})\Big) = \mathcal{D}(\boldsymbol{z}^{(i)}) \approx \boldsymbol{f}^{(i)}. \tag{4.1}$$

The denoiser is a UNet $\mathbb{U}_\theta$ that predicts the noise added during the forward diffusion process in order to recover clean latents. Its inputs are:

$$\mathbb{U}_\theta\Big(\hat{\boldsymbol{z}}_t, \boldsymbol{z}^{(0)}_{\text{pad}}, \boldsymbol{c}^{(0)}, t\Big), \tag{4.2}$$

where $\hat{\boldsymbol{z}}_t$ are the predicted noisy latents at timestep t, $\boldsymbol{z}^{(0)}$ is the latent of the initial frame, $\boldsymbol{z}^{(0)}_{\text{pad}}$ is $\boldsymbol{z}^{(0)}$ tiled N times along the temporal dimension, and $\boldsymbol{c}^{(0)}$ is the CLIP [43] embedding of the initial frame. The CLIP embedding provides a semantically rich representation aligned across image and text modalities.

Ctrl-V augments SVD with a ControlNet module to incorporate explicit spatial control signals. Alongside the original SVD inputs, the ControlNet receives encoded bounding box sequences $\boldsymbol{b}$ that specify object locations for each frame. These bounding boxes are processed through a dedicated U-Net adapter in the ControlNet branch. The resulting control features are injected into the decoder layers of the SVD UNet via residual connections after passing through a zero-initialized 1×1 convolution, ensuring they guide generation without dominating it initially (refer to Section 2.3 for more information on how the ControlNet adapter works).

Because the control signal is defined in pixel space before encoding, the architecture maintains strong spatial alignment between the conditioning input and the generated video. This enables fine-grained control not only over *where* objects appear, but also over *how* their identity and motion evolve; supporting object-specific animation under user control.

For the purposes of this thesis, the most important aspects of Box2Video are: (i) its latent-space video diffusion backbone (SVD), which allows single-frame conditioning; and (ii) the ControlNet-based spatial conditioning pathway, which enables bounding box-guided motion synthesis. Other implementation details (e.g., attention mechanisms, temporal modeling blocks) are beyond the scope of this thesis but follow the original SVD architecture [7].

4.3.3 Performance and Output Quality

Table 4.1 reports the quantitative evaluation of Ctrl-V under various ablations, alongside comparable baselines. The "Stable Video Diffusion Baseline" is the off-the-shelf SVD model, while the "Stable Video Diffusion Fine-tuned" variant is trained on the same nuScenes [44] dataset as Ctrl-V. The "BBox Generator + Box2Video" configurations represent the full Ctrl-V pipeline: starting from a single initial frame and the final ground-truth bounding box frame, the model predicts intermediate bounding boxes and then synthesizes video. The "1-to-1" and "3-to-1" variants differ in how many initial ground-truth bounding box frames are given (1 or 3). In contrast, the "Teacher-forced Box2Video" variant bypasses bounding-box prediction entirely by feeding the ground-truth box trajectories directly to the Box2Video stage. For reference, we also include DriveGAN [45] and DriveDreamer [46], two vehicle-oriented generative models, as additional baselines.

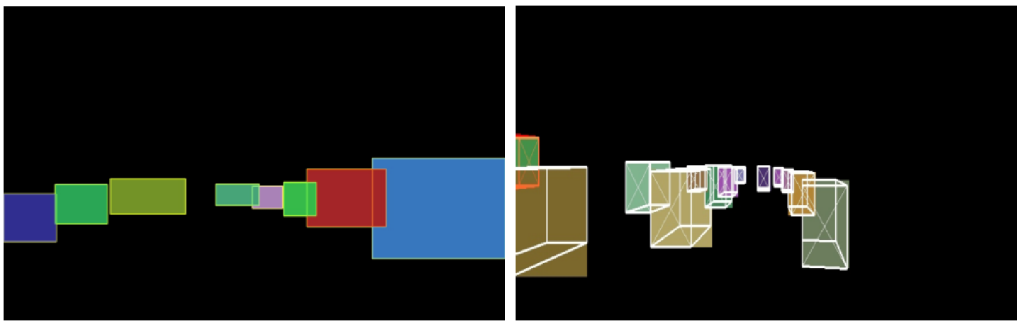
Pipeline	**#BBs**	**FVD↓**	**LPIPS↓**	**SSIM↑**	**PSNR↑**
Stable Video Diffusion Baseline	None	1179.4	0.5004	0.2877	13.31
Stable Video Diffusion Fine-tuned	None	316.6	0.2730	0.4787	18.58
Ctrl-V: BBox Generator + Box2Video	1-to-1	285.3	0.2647	0.5050	18.93
Ctrl-V: BBox Generator + Box2Video	3-to-1	**235.0**	0.2235	0.5500	20.33
Ctrl-V: Teacher-forced Box2Video	All	235.5	**0.2104**	**0.5705**	**23.36**
DriveGAN	None	390.8	-	-	-
DriveDreamer	All	340.8	-	-	-

Table 4.1 **Comparing the quality and diversity of the generated video models.** The evaluated sample videos have a resolution 256×410 and consist of 11 frames at 4 Hz. The "#BBs" column reports the number of ground-truth input bounding box frames used by the generation pipelines. "None" indicates that no ground-truth frames are used, while "All" indicates that all ground-truth bounding box frames are utilized. If "#BBs" is n-to-m, then it represents the number of initial bounding box frames used by the pipeline is n and the number of final bounding box frames used by the pipeline is $m = 1$.

We evaluate video quality using four standard metrics: FVD, LPIPS, SSIM, and PSNR. LPIPS, SSIM, and PSNR compare generated and ground-truth frames either in feature space (LPIPS) or pixel space (SSIM, PSNR), capturing perceptual fidelity, structural consistency, and reconstruction accuracy, respectively. FVD measures distribution-level similarity between real and generated videos, capturing both spatial and temporal coherence (see Background section 2.4 for more details on all these metrics). For SVD and Ctrl-V, these metrics are computed over 200 generated samples; FVD values for DriveGAN and DriveDreamer are taken from the original publications.

Across all metrics, Ctrl-V consistently outperforms the other methods, demonstrating the benefit of explicit trajectory conditioning. The teacher-forced configuration yields the strongest LPIPS, SSIM, and PSNR scores, indicating that perfect bounding-box inputs slightly improve visual fidelity. However, the gap with the full Ctrl-V pipeline is small, showing that the bounding-box prediction stage introduces only minor degradation. Notably, the "3-to-1" variant achieves the best FVD score overall, suggesting that providing more initial trajectory context leads to improved temporal realism. These results confirm that Ctrl-V 's design enables high-quality, controllable driving video generation that generalizes beyond the capabilities of standard video diffusion models.

4.4 Bbox Generator



(a) **2D bounding box frame** (b) **3D bounding box frame**

Figure 4.3 **Examples of bounding box frames in the Ctrl-V framework**.

While the Box2Video model forms the core of Ctrl-V 's capability to generate realistic visual scenes, it relies on the structured inputs provided by the Bbox Generator. The primary purpose of the Bbox Generator within the Ctrl-V pipeline is to serve as a controllable and interpretable interface between scene-level semantics and the low-level visual synthesis performed by Box2Video. Rather than generating raw pixel videos from scratch, Ctrl-V decomposes the task into two stages, where the first stage produces bounding box sequences that define multi-object motion in a visually encoded form, which are then translated by Box2Video into photorealistic video frames.

4.4.1 Method

In this setting, the Bbox Generator is responsible for producing plausible multi-agent trajectories over time, capturing not only the positions and velocities of vehicles and pedestrians, but also their identities, semantic classes, and orientations. A central design choice in Ctrl-V

is to render these trajectories directly into RGB video frames, treating bounding boxes as first-class visual entities. This visual representation embeds each object's properties—such as track ID (via consistent color), class type (via border color), and orientation (via directional markings)—within the frame itself. This allows the diffusion model to operate over structured visual inputs rather than tokenized coordinates, benefiting from well-established inductive biases in image generation. Bounding boxes can be represented as 2D boxes for simplicity or 3D bounding boxes for more detailed definition in 3D space (including orientation and depth size). Figure 4.3 shows examples for 2D and 3D bounding box frames.

Technically, the Bbox Generator is implemented as a video diffusion model fine-tuned to synthesize bounding box renderings over time. As shown on the left side of Figure 4.1, the model uses a SVD based on a U-Net video backbone (just like the Box2Video model) trained in a denoising diffusion framework, where noisy sequences of bounding box frames are progressively refined into clean, coherent motion. Conditioning information, such as scene layout or context frames, can be injected through cross-attention or latent concatenation mechanisms, steering the generation toward globally consistent trajectories. The resulting sequences are both semantically dense and visually interpretable, enabling direct control over agent behaviour in the final video output produced by Box2Video. The four inputs to the model are: the encoded "video" of bounding boxes with t levels of noise added $\hat{\boldsymbol{b}}_t$, the encoded initial bounding box frame(s) $\boldsymbol{b}^{(0)}$, the encoded final bounding box frame $\boldsymbol{b}^{(N-1)}$, and the encoded initial video frame $\boldsymbol{z}^{(0)}$. During training, the denoiser model learns to predict the noise added in $\hat{\boldsymbol{b}}_t$, allowing the recovery of $\boldsymbol{b}$ after subtracting the predicted noise from $\hat{\boldsymbol{b}}_t$. The training signal is therefore encouraging the reconstruction of the input bounding box sequence, however this training scheme enables the generation of novel and diverse bounding box trajectories conditioned on the first (or few first) and last bounding box frames.

4.4.2 Comparison to Autoregressive Baselines

To evaluate both the performance and simplicity of our bounding box generator model, we implemented a baseline method inspired by the Trajeglish model [33], which we refer to as the "Trajeglish-Style" model.

The original Trajeglish model employs a discrete sequence modeling approach to represent multi-agent road-user trajectories. Vehicle trajectories are modeled as a discrete sequence of actions, where each action corresponds to a token that describes the agent's relative displacement. The model utilizes a GPT-like encoder-decoder architecture, taking as input the agents' initial timestep information and map objects, and outputting a distribution over possible displacement actions. Trajeglish operates in BEV and achieved state-of-the-art (SOTA)

Dataset	Method	maskIoU↑	maskP↑	maskR↑
KITTI	BBox Generator	**.795** ± .112	**.881** ± .082	**.884** ± .078
	Trajeglish-Style	.491 ± .164	.622 ± .173	.691 ± .175
BDD	BBox Generator	**.647** ± .176	**.784** ± .150	**.783** ± .156
	Trajeglish-Style	.373 ± .185	.454 ± .206	.686 ± .193
nuScenes	BBox Generator	**.827** ± .150	**.892** ± .120	**.906** ± .099
	Trajeglish-Style	.448 ± .194	.554 ± .213	.695 ± .196

Table 4.2 **Bbox Generator performance by comparing real and generated bounding boxes**. We condition on 3 initial bounding box frames and 1 final bounding box frame. The last three columns show evaluations on the entire generated bounding box sequence, measuring the alignment scores between our generated bounding box generations and ground-truth labels. "BBox Generator" is our method and "Trajeglish-Style" is a baseline inspired from [33].

performance on the 2023 WOMD Sim Agents Benchmark.

To adapt this method to our problem, we made several modifications. Our "Trajeglish-Style" model focuses on predicting 2D bounding boxes from the ego perspective (POV). Unlike the BEV representation, these bounding boxes can overlap in 3D space and are subject to resizing during rollout. Additionally, the boxes can dynamically enter and exit the frame at any point in the sequence.

To evaluate the quality of our bounding box generations, we create mask images for both the ground-truth and generated bounding box sequences. The mask images are generated by converting the bounding box frames into binary masks. We then calculate the generated averaged mask intersection over union (maskIoU) scores, averaged mask precision (maskP) scores, and averaged mask recall (maskR) scores against the ground-truth bounding box masks. To assess our bounding box trajectories, we average the results over 200 generated samples. We use 3 different datasets for evaluation, where the models have been finetuned exclusively on these datasets for these experiements: KITTI [47], BDD100k [48], and nuScenes [44]. Results for Ctrl-V's Bbox predictor are compared with those of the "Trajeglish-Style" model in Table 4.2.

While the original Trajeglish model achieved SOTA performance in its intended task of predicting BEV agent trajectories, we do not claim that our "Trajeglish-Style" model reaches SOTA for the current task of ego POV 2D bounding box trajectory prediction. In fact, we consider this task to be significantly more challenging and less suited to the modeling approach of Trajeglish, which was not originally designed for the complexities of predicting

the motion of bounding boxes in the ego perspective. In contrast, the video diffusion-based method used in Ctrl-V, it suffers from limitations in spatial smoothness, temporal coherence, and scalability to complex scenes with multiple interacting agents.

4.5 Towards Crash Video Generation

While Ctrl-V enables high-quality and controllable video synthesis from bounding box trajectories, its training is primarily focused on nominal, everyday driving behavior. As such, it does not directly capture high-risk dynamics such as collisions, sudden stops, or multi-agent impacts. We will continue to progress towards our goal of generating safety-critical driving scenarios in the following chapters. Namely, in Chapter 5 we will stress test the Ctrl-V framework by attempting to generate car crashes videos, followed with the introduction of Ctrl-Crash in Chapter 6, a crash-oriented extension of Box2Video designed specifically to model and generate realistic driver-perspective crash videos.

CHAPTER 5 BEV2POV: BRIDGING BEV AND VIDEO

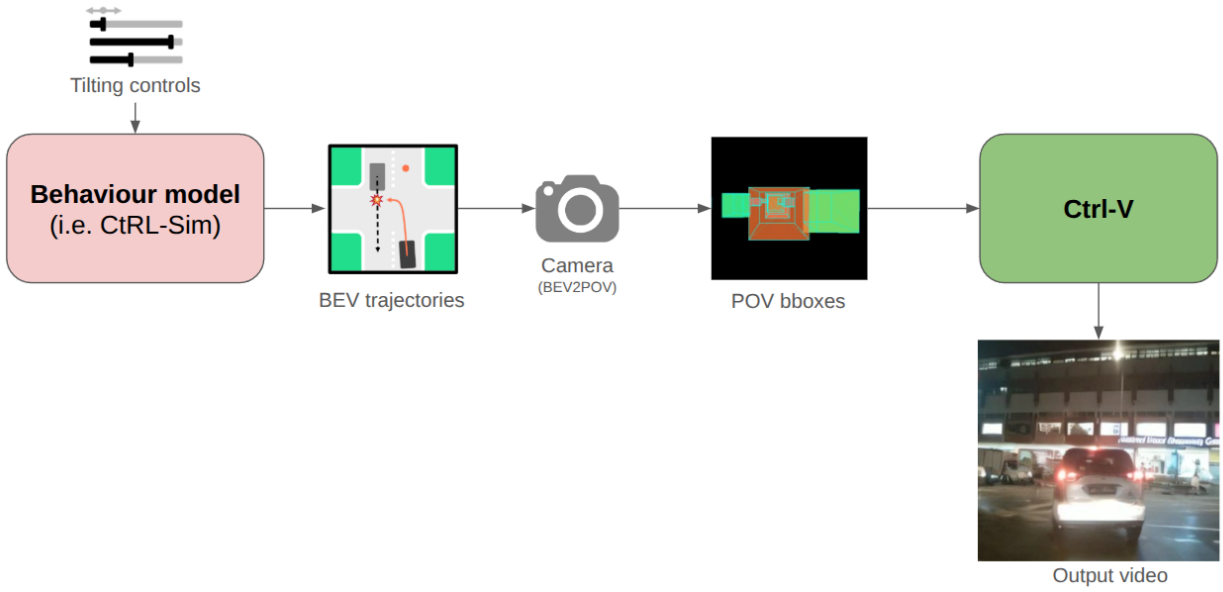
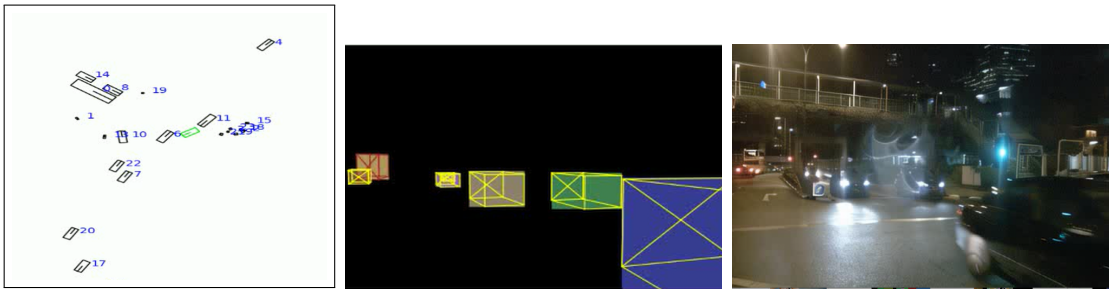


Figure 5.1 **BEV2POV pipeline concept.** Suggested pipeline for connecting a BEV-based behavior model—such as CtRL-Sim —with Ctrl-V, a video generation model conditioned on bounding boxes in first-person perspective. This is enabled via a "BEV2POV" transformation that projects BEV trajectories into the ego-centric camera view.

In the preceding chapters, we introduced two complementary paradigms for scenario generation in autonomous driving. CtRL-Sim operates in BEV, leveraging structured policy learning to generate behaviorally rich multi-agent trajectories. In contrast, Ctrl-V enables video generation in the ego-agent's POV through object-level motion control using bounding box trajectories. While these approaches differ in representation and output modality, both aim to address the same overarching challenge: creating controllable and realistic scenarios for AV testing and training.

In this chapter, we propose a simple yet powerful conceptual and technical bridge between these paradigms, which we refer to as BEV2POV. The key idea is to take behaviorally-defined scenarios, such as those generated by CtRL-Sim, scripted manually in BEV, or from ground-truth data, and transform them into the format required for video generation using Ctrl-V. This enables us to repurpose BEV trajectory-based simulation frameworks to synthesize visual scenes, without modifying the underlying video diffusion model. See Figure 5.2 for visual examples of the different representations discussed in this chapter.



(a) BEV (b) Ego POV (c) Generated Frame

Figure 5.2 **Examples for the different representations and point of views for BEV2POV. (Left)**: A BEV frame from the nuScenes dataset showing agents with 2D bounding boxes with the ego agent in green. **(Middle)**: The equivalent frame in the ego agent POV with other agents represented with 3D bounding boxes. **(Right)**: Output frame from Ctrl-V conditioned on the 3D bounding box frame (and an image frame at a previous timestep).

5.1 From BEV Trajectories to Ego-View Conditioning

At the core of the BEV2POV pipeline is a geometric transformation that projects 2D agent trajectories from the BEV coordinate system into the egocentric camera view of the autonomous vehicle. A scenario generator such as CtRL-Sim produces a set of agent trajectories $\{\tau_i(t)\}$, where each trajectory is defined by a sequence of agents' positions and orientations over time in a global top-down coordinate frame.

To convert these into input compatible with Ctrl-V, we first lift these 2D bounding boxes into 3D cuboids. Each box is assigned a height according to the semantic class of the object (e.g., car, pedestrian, bus, etc.), using a class-specific normal distribution estimated from real-world data (i.e., from the nuScenes dataset). For example, a vehicle's height h may be drawn from $\mathcal{N}(\mu_{\text{car}}, \sigma^2_{\text{car}})$, where μ_{car} is the mean vehicle height in the training dataset. This allows us to reconstruct an approximate 3D bounding box for each agent:

$$B_t^{(i)} = \left(x_t^{(i)}, y_t^{(i)}, z_t^{(i)}, w^{(i)}, l^{(i)}, h^{(i)}, \theta_t^{(i)}\right) \tag{5.1}$$

where (x, y, z) denotes the center position, (w, l, h) the dimensions, and θ the orientation for the agent i at timestep t.

To convert a 3D bounding box in world coordinates to the 2D image plan of the ego's POV, we focus on the problem of transforming each one of the 8 corners that defines it from a 3D point to a 2D point on the image plane. Let $\mathbf{p}_w = [x_w, y_w, z_w]^\intercal$ be a 3D point in world

coordinates.

To convert a 3D point to a 2D image-plan point—relative to a camera's point of view—we can leverage some matrix transformations in a process we refer to as camera projection for a pinhole camera model [49]. First, we define the camera intrinsic matrix $\mathbf{K} \in \mathbb{R}^{3\times3}$ as the following matrix:

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_Y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

where f_x and f_y are the focal length components for the camera in pixel units which will normally be the same value and can be expressed simply as the focal length f of the camera ($f_x = f_y = f$). c_x and c_y represents the coordinates to the optical center or "principal point offset", which represents how far the camera's image center is shifted from the actual center of the image sensor. In other words, this tells you the coordinates (in pixels) of the image center from the top-left corner of the image, which is normally the exact center of the image: $c_x = W/2, c_y = H/2$ where W and H are the image width and height in pixels. Finally, s is the axis skew which causes a shear distortion ($s = 0$, here).

The camera intrinsic matrix $\mathbf{K}$ is useful for points centered in the camera's coordinate frame. To map a point in world coordinates to camera-centered coordinates we must make use of the camera extrinsic matrix $[\mathbf{R} \mid \mathbf{t}] \in \mathbb{R}^{3\times4}$:

$$[\mathbf{R} \mid \mathbf{t}] = \left[\begin{array}{ccc|c} R_{1,1} & R_{1,2} & R_{1,3} & t_0 \\ R_{2,1} & R_{2,1} & R_{2,3} & t_1 \\ R_{3,1} & R_{3,1} & R_{3,3} & t_2 \end{array}\right] \tag{5.3}$$

The camera extrinsic matrix is composed of a rotation matrix $\mathbf{R}$ augmented with a translation vector $\mathbf{t}$ which describes how to transform world coordinates to camera coordinates:

$$\mathbf{p}_c = [\mathbf{R} \mid \mathbf{t}]\bar{\mathbf{p}}_w \tag{5.4}$$

where $\mathbf{p}_c = [x_c, y_c, z_c]^\intercal$ is a 3D camera-centered point and $\bar{\mathbf{p}}_w = [x_w, y_w, z_w, 1]^\intercal$ is a 3D world-centered augmented point.

We now have everything we need to convert a point in 3D world space to the 2D camera plane in homogenous coordinates like so:

$$\tilde{\mathbf{x}}_s = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]\bar{\mathbf{p}}_w \tag{5.5}$$

where we obtain the 2D homogenous point $\tilde{\mathbf{x}}_s = [\tilde{x}_s, \tilde{y}_s, \tilde{w}_s]^\intercal = \tilde{w}_s[x_s, y_s, 1]$. Here we are interested in x_s and y_s, which are the 2D coordinates of the point on the image plane, note that $\tilde{w}_s$ is a scaling factor resulting from the geometric transformation done in homogenous coordinates. By converting the 8 corners of a 3D bounding box to the camera's image plane using Equation 5.5, we can render the 3D bounding boxes as they are visible to the ego agent (such as seen in Figure 5.2b).

5.2 Proof of Concept: BEV-Guided Crash Scenario Generation

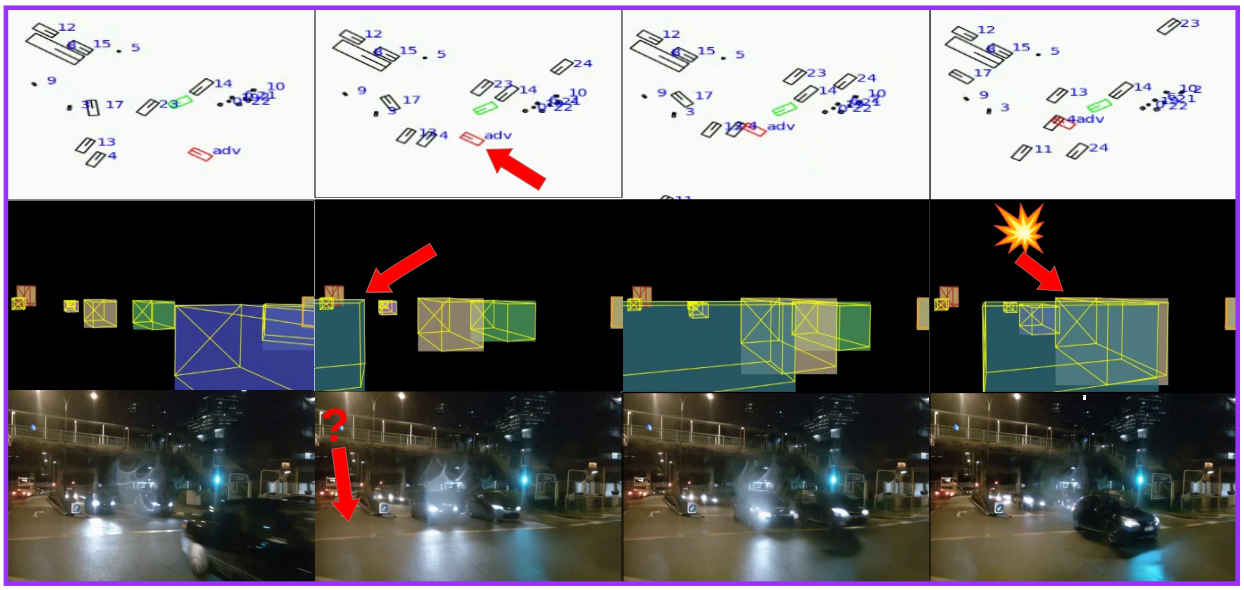


Figure 5.3 **BEV2POV Crash scenario example:** Attemping to generate a car crash video using Ctrl-V and conditioned on a BEV scenario converted to ego POV 3D bounding boxes via BEV2POV. Each row shows a different modality at 4 different timesteps. **(Top row)**: BEV trajectory from nuScenes dataset altered to simulate a crash, where the ego agent is shown in green and a new adversarial ("adv") agent in red was manually added to cause a crash with an agent directly in front of the ego agent. **(Middle row)**: 3D bounding boxes projected to the POV of the ego agent via the BEV2POV process. **(Bottom row)**: The generated frames from Ctrl-V conditioned on the 3D bounding box sequence. We notice that the adversarial agent does not appear from the left side (red arrows, second column) and the other car to be involved in the crash stops in the middle of the intersection, but does not crash.

One key benefit of BEV2POV is its ability to enable experimentation with scene structure

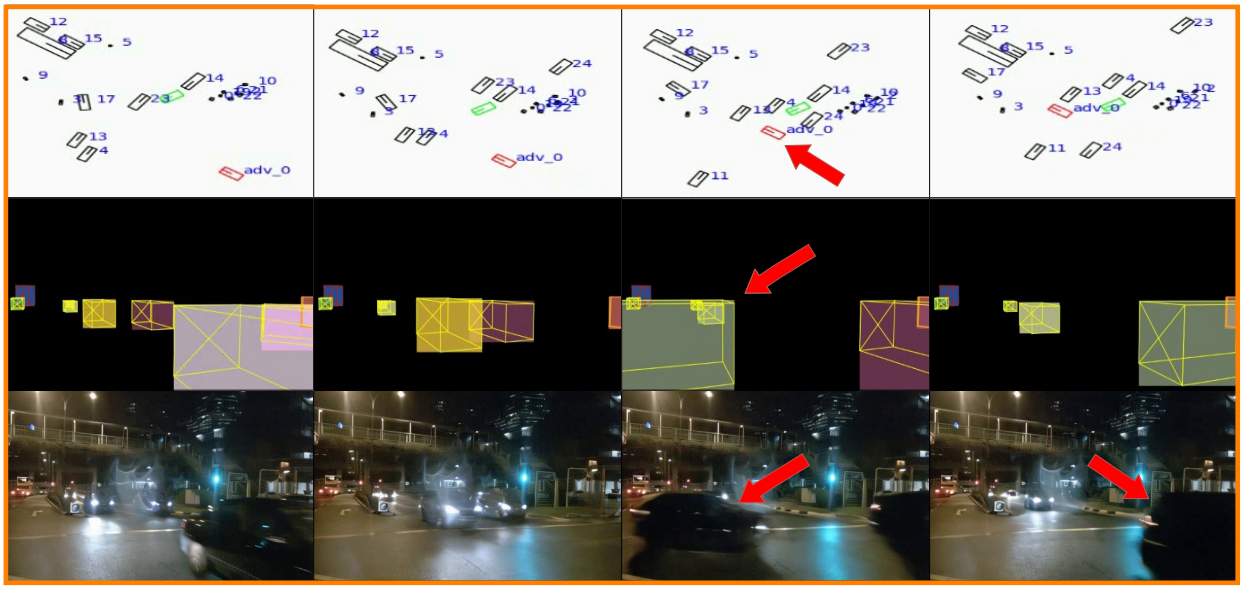


Figure 5.4 **BEV2POV Crash-free scenario example alternative:** *Crash-free* scenario offering an alternative to the crash scenario depicted in Figure 5.3 where the adversarial vehicle that is added manually clears the intersection later and does not collide with another car. This shows the ability of Ctrl-V to generate new agents using the BEV2POV pipeline. **(Top row)**: BEV trajectory from nuScenes dataset altered with a new adversarial ("adv_0") agent in red that drives across the intersection without colliding with any cars (as opposed to colliding with the agent "4" as in Figure 5.3). The ego agent is shown in green. **(Middle row)**: 3D bounding boxes projected to the POV of the ego agent via the BEV2POV process. **(Bottom row)**: The generated frames from Ctrl-V conditioned on the 3D bounding box sequence. We notice that the adversarial agent that is not involved in a crash appears from the left side as expected and that all cars continue to drive normally and as directed.

and agent behavior in a low-dimensional, interpretable space. BEV trajectories are easier to generate, edit, and manipulate than full POV videos, which makes them an appealing interface for crafting novel driving scenarios. In particular, we used this framework as a proof of concept to explore whether Ctrl-V could generate crash events from BEV-defined interactions.

To this end, we constructed a number of synthetic BEV crash scenarios derived from the ground-truth nuScenes dataset. These included inserting new adversarial agents into the trajectories, modifying existing agent paths to create future collisions, or generating entirely new multi-agent scenes with intended impact events. An example of this pipeline and the resulting failure case is shown and described in Figure 5.3, where we inserted a new vehicle into a ground-truth nuScenes scene such that it would follow a simple linear trajectory and decelerate suddenly upon overlap with another agent, resulting in a programmed crash. This

adversarial agent was not part of the original log, but was positioned to collide with a vehicle directly visible by the ego vehicle. The BEV trajectory is projected into the ego view and rendered into a 3D bounding box sequence, followed by the output sequence from Ctrl-V which fails to depict the expected crash.

It is plausible to think that the failure case shown in Figure 5.3 could be due to Ctrl-V struggling to generate a novel agent that does not exist in the initial conditioning image by having it appear from out of frame. However, we outrule this hypothesis by showing an alternative scenario in Figure 5.4 where the novel "adversarial" agent that was programmed to crash into the other car in Figure 5.3, this time, traverses the intersection at a later timestep such that it does not cross paths with the other vehicle. In this situation, we can see that Ctrl-V is capable of generating this new vehicle. We conducted other tests similar to these using different scenarios, different agent types, different types of collisions (t-bone, rear-end, head-on, etc...) and even other events like near misses, reckless swerving and sudden braking. One thing was consistent: if the conditioning instructed the model to generate some rare/dangerous behaviour such as a crash, it would systematically refuse to do so. The generated video would diverge dramatically: objects would vanish prematurely, remain static, or the video would simply depict uneventful driving.

Suprisingly enough, it is not only Ctrl-V that lacks the ability to generate a convincing car crash, even state-of-the-art diffusion-based video models, such as OpenAI Sora [9], Nvidia Cosmos [8], and DeepMind Veo3 [10] fail to generate plausible crash scenarios. We tested these three models with a few generation samples, and though they tend to generate high definition and smooth video, they consistently fail to generate physically plausible crashes (if any crash at all). We show examples in Figure 5.5. In the top row of the figure we show a sample result from Nvidia Cosmos-Predict1-7B-Text2World that was instructed with the following text prompt: *"On a highway two cars collide at very fast speeds head-on"*. This produces a highly implausible scene, where the car on the left suddenly starts to levitate from its rear-end and a cloud of smoke resembling an explosion appears, followed by disjointed fragments of torn metal emerging from the ground that transforms into a dark vehicle rushing towards the camera. The Nvidia Cosmos model suite has been specifically trained for physics-aware generation, but still fail to generate a car crash. Alternatively, we show a sample from OpenAI's Sora video model in Figure 5.5 in the middle row. The Sora video model was given the prompt: *"At an intersection two cars collide with each other at full speed resulting in a crash"*. The resulting video shows a car that spins erratically, changes shape and direction, and produces visible artifacts—culminating in a pile of twisted metal and glass. Finally, we show an example using DeepMind's Veo3 in the bottom row of Figure 5.5. Perhaps the closest to depicting a car crash among the three when looking at individual frames, but when

Figure 5.5 **State-of-the-art video diffusion models instructed to generated a car accident: (Top row)**: Nvidia Cosmos (7B Text2World), **(Middle row)**: OpenAI Sora, **(Bottom row)**: DeepMind Veo3.

watching the video as a whole it becomes clear that the events are completely unrealistic. In the generated video, we see two still vehicles very close to each other that start accelerating, then we see debris and smoke as the cars slowly move towards each other, then the front of the cars phase through each other and reveal completely undamaged cars, and finally the red car drives away from the scene unscathed.

These tests strongly suggests that many diffusion-based video generation models—including Ctrl-V, Cosmos, Sora, and Veo3—are not capable of generating realistic car crashes. On top of this, they are not designed out-of-the-box for precise control as they are generally conditioned on text only. In the case of Ctrl-V, the model was trained on normal driving data, which may explain why it lacks the capacity to synthesize out-of-distribution (OOD) visual events such as crashes, even when the input control signal explicitly defines them. The absence of such events in the training distribution likely causes the diffusion model to fall back to safe or visually conservative outputs, filtering or ignoring trajectories that are deemed implausible by the learned prior.

5.3 Motivation for Crash-Specific Video Generation

The limitations of Ctrl-V (and SOTA models) in handling crash scenarios motivates the development of a specialized video generation model trained explicitly to model and synthesize dangerous or rare driving events. The experiments shown in this chapter offer strong evidence that even SOTA video diffusion models do not generalize to safety-critical edge cases, and that new architectures, training strategies, and datasets are needed to address this gap.

In the remainder of this thesis, we introduce Ctrl-Crash, a controllable video generation model designed specifically for crash simulation. Ctrl-Crash builds on the bounding box conditioning framework of Ctrl-V but is trained on crash-focused video data and incorporates additional mechanisms to better handle the creation of safety-critical driving scenarios.

CHAPTER 6 CTRL-CRASH: CONTROLLABLE CRASH GENERATION

This chapter draws from the author's first-author publication, Ctrl-Crash [50], and represents the main contribution of this thesis. This chapter builds upon and is motivated by the work presented in the previous chapters.

6.1 Introduction

Despite the seemingly high quality video generation results produced by the latest wave of generative models, they systematically fail to generate physically plausible crashes and tend to break down, yielding highly implausible imagery. As an example, we looked at different state-of-the-art models in the Chapter 5, namely: Nvidia Cosmos, OpenAI Sora, and DeepMind Veo3 (Figure 5.5).

To address the scarcity of realistic and controllable crash data, this chapter introduces Ctrl-Crash, a generative video framework designed to synthesize realistic car crash scenarios from a single initial frame. Unlike classical graphics-based physics simulators, Ctrl-Crash operates with inputs and outputs in the pixel domain using a data-driven video diffusion model. It enables fine-grained control over the crash generation process through two forms of conditioning: (1) spatial control signals, provided as bounding box trajectories for road users, and (2) semantic intent signals, specified as five discrete crash types. Together, these inputs allow the system to simulate plausible crash outcomes, modify agent behavior, and explore alternate versions of the same scenario—providing the ability to ask and answer counterfactual questions such as: *How would the outcome change if a vehicle braked earlier, or if a different type of collision occurred?*

Compared to traditional physics-based rendering pipelines, which rely on game engines or explicit simulation of collision dynamics, Ctrl-Crash offers several key advantages. Physics-based approaches often require extensive human labor for scenario design, high-fidelity asset modeling, and rendering, while still struggling to match the visual realism of real-world imagery. In contrast, data-driven methods such as Ctrl-Crash learn directly from real video data, enabling visually realistic and semantically diverse outcomes. While they may not model physical laws explicitly, these models implicitly capture behavioral and causal patterns observed in real driving environments, offering a powerfully scalable alternative for simulating long-tail safety-critical events.

By supporting explicit control over agent motion and crash types, Ctrl-Crash offers a scalable

and flexible tool for simulating safety-critical scenarios in autonomous driving. It aims to bridge the gap between data-driven video synthesis and high-level behavioral intent modeling, contributing toward more diverse and causally informative simulations that are essential for robust AV training, testing, and evaluation.

This work makes the following key contributions toward advancing safety-critical scenario simulation for autonomous driving:

1. **A controllable generative video model for crash simulation**: Ctrl-Crash enables the synthesis of realistic dashcam-style accident videos. It incorporates both spatial (bounding box trajectories) and semantic (crash type) conditioning signals to direct the outcome of a scene. The model achieves state-of-the-art performance among diffusion-based approaches for accident video generation, as evaluated through standard quantitative metrics (e.g., FVD) and human assessments of visual realism and physical plausibility.

2. **Support for counterfactual reasoning through fine-grained control**: The proposed framework allows multiple plausible crash outcomes to be generated from the same initial scene by varying the control inputs. This enables counterfactual simulations that can explore how subtle changes in intent or agent trajectories might influence the evolution of a crash—providing valuable insight for safety-critical reasoning and failure analysis.

3. **A complete crash video processing pipeline**: To support training and evaluation, a pipeline was developed to filter dashcam videos and extract bounding box trajectories of road users. Based on this, a curated extension of the MM-AU dataset [51] is constructed with bounding box annotations, along with held-out test sets from the Car Crash Dataset RUSSIA [52] and BDD100k [48]. These resources are released alongside tools to facilitate future research in crash simulation and controllable video generation.

The remainder of this chapter presents the design and evaluation of Ctrl-Crash, a data-driven framework for controllable car crash video generation. We describe the overall architecture 6.2, conditioning mechanisms 6.3, training strategy 6.4, our extension of classifier-free guidance for fine-grained control 6.5, data processing pipeline 6.6, along with quantitative 6.7 and qualitative 6.8 results and evaluation for Ctrl-Crash.

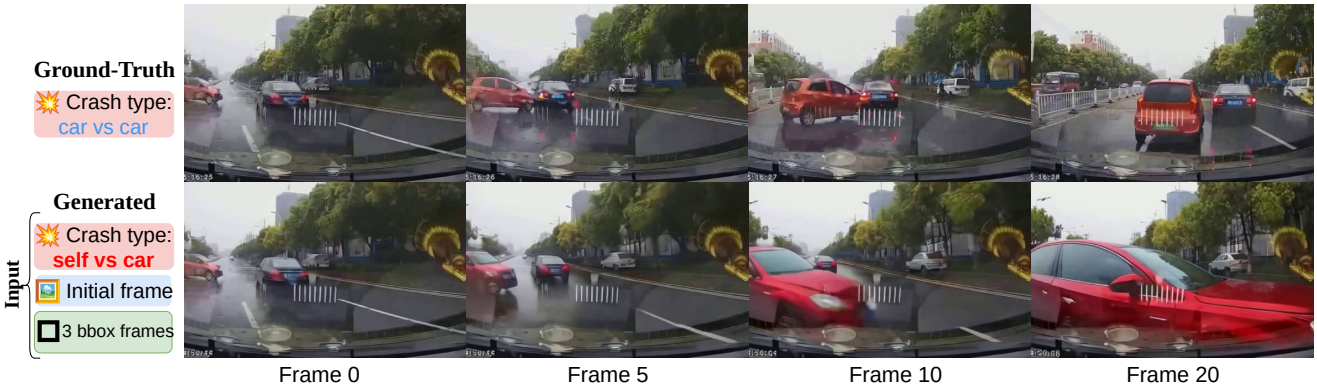


Figure 6.1 **Counterfactual Crash Generation**: this diagram demonstrates the ability of our model to generate counterfactual crashes while beginning from the identical initial frame. **(Top row):** a ground truth accident between two vehicles other than the ego-vehicle, where the red car hits the rear of the blue car and spins into the lane in front of the ego-vehicle. **(Bottom row)**: the model generates an alternative accident involving the ego-vehicle. In this alternative future the red car avoids the blue car but turns into the path of the ego-vehicle leading to the crash.

6.2 Overview

Ctrl-Crash, is a controllable video diffusion framework designed to generate realistic car crash scenarios from a single initial frame, guided by both spatial and semantic control signals. Ctrl-Crash builds on Ctrl-V [41], a framework for generating videos from rendered bounding box trajectories (presented in Chapter 4), by extending its capabilities to crash-specific scenarios, offering richer control and greater flexibility. Specifically, we incorporate a new semantic control signal representing crash type and introduce a refined training procedure to handle partial and noisy conditioning.

Our method follows a two-stage training pipeline. In the first stage, we fine-tune the Stable Video Diffusion (SVD) model on a curated version of the MM-AU [51] crash video dataset to improve its ability to synthesize dynamic and physically plausible driving and accident scenes. In the second stage, we train a ControlNet module to inject conditioning information in two forms: (1) bounding box sequences representing road user motion across time, and (2) discrete crash type labels encoding high-level semantic intent. To ensure generalization to incomplete or noisy control, we introduce a curriculum-based random masking strategy that progressively masks out parts of the control inputs during training. Masked bounding box frames are replaced by a learnable embedding that preserves scene plausibility. We further extend classifier-free guidance to support independent scaling of each control modality, enabling nuanced and flexible control at inference. Unconditional noise predictions are obtained from the pretrained base model for improved generation diversity and stability (see Section 6.4).

Ctrl-Crash supports three task settings, each enabled by varying the available control signals:

1. ***Crash Reconstruction***: Given an initial image, full bounding box sequence, and a crash type, the model reconstructs a consistent video combining the visual context of the initial frame with agent motion derived from the bounding boxes.

2. ***Crash Prediction***: Given the initial frame and either none or a few initial bounding box frames (e.g., 0–9), the model predicts the future motion of agents in a way that aligns with the target crash type.

3. ***Crash Counterfactuals***: Extending the prediction task, this mode varies the crash type signal while keeping other inputs fixed, enabling the generation of multiple plausible outcomes for the same scene—supporting counterfactual safety reasoning. Figures 6.1 and 6.7 illustrate this counterfactual capability. We use the term "counterfactual" here in an informal sense, not to imply a formal causal model, but rather to describe alternate plausible outcomes from the same initial conditions. Rather it serves as a valuable capability for safety analysis and planning under alternative futures.

6.3 Conditioning Signals

Ctrl-Crash generates videos from three complementary conditioning signals that guide both the visual realism and semantic plausibility of the crash scenario:

6.3.1 Initial Frame (Scene Context):

The initial image provides the visual grounding for generation, capturing the appearance, layout, and environment of the driving scene before any dynamic event unfolds. It is encoded using a pretrained VAE and additionally a CLIP encoder, and fed into the base diffusion model as context. The initial image strongly influences scene appearance and plausibility, and serves as the starting point for generating temporally coherent crash evolutions. For example, the initial image will set the environment (e.g., an intersection, a busy highway, a calm country road, etc.) and frame the location of the ego agent (the observer) and other participants within it. All agents present in the frame will be animated as well as any additional agents that may enter the frame throughout the video. Note that this frame will not be static in the video as the ego agent will also be controlled and our point of view in the environment will evolve.

6.3.2 Bounding-Box Trajectories (Spatial Control):

Road users are represented with 2D bounding boxes rasterized into RGB control frames. These provide spatial guidance regarding each object's motion, size, and location over time. Each control frame encodes the complete set of bounding boxes for a single timestep. Each bounding box is color-coded to encode both its unique track ID (via fill color) and its object class (via border color), enabling the model to distinguish agents across frames. The control frame has the same resolution as the input video (e.g., 512×320) and can represent any number of agents per frame. If depth information is available, bounding boxes that are farthest away are drawn first with closer ones drawn over, if no depth information is available overlapping boxes are drawn in arbitrary order. These control frames are encoded using the same pretrained VAE as the initial image and processed by a ControlNet and injected into the denoising process, directing the motion of agents in the generated video. An example of a bounding-box control frame, alongside its corresponding real image, is shown in Figure 6.2.

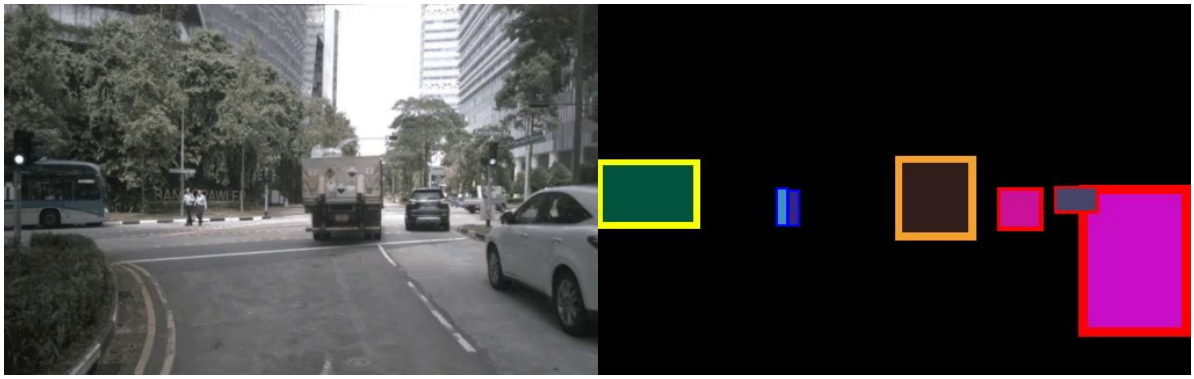


Figure 6.2 **Ctrl-Crash bounding box frame example**. **(Left)**: example frame from a driving video (from BDD100k dataset). **(Right)**: associated bounding box frame conditioning generated from our pipeline. Road users are represented as 2D bounding boxes with unique fill colors representing their track ID and specific border colors representing their class.

Track ID Encoding (Fill Color) Each bounding box is filled with a unique RGB color derived from its object's persistent track ID. The RGB values are deterministically generated via a hashing function to ensure consistency across frames. RGB values vary within $[50, 255]$ for all three color channels. This allows the model to temporally link the same agent across timesteps and learn coherent motion patterns. The use of color fills avoids the need for explicit ID embeddings and leverages the spatial structure of the image.

Class Encoding (Border Color) To distinguish between different types of road users (e.g., cars, trucks, buses, pedestrians, cyclists, etc.), we draw a thick border around each bounding box in a class-specific color. These colors are chosen from a fixed palette (see Table 6.1), and the mapping between semantic classes and RGB border values is consistent across all training data . This helps the model differentiate object behaviors by class, which is particularly useful in crash prediction (e.g., trucks tend to behave differently from bicycles).

Class	Border Color	RGB values
person	Blue	(0, 0, 255)
car	Red	(255, 0, 0)
truck	Orange	(247, 162, 44)
bus	Yellow	(250, 255, 2)
train	Green	(0, 255, 0)
motorcycle	Purple	(204, 153, 155)
bicycle	Pink	(255, 209, 22)

Table 6.1 **Class encoding color scheme for bounding box border color**

6.3.3 Crash Type Label (Semantic Control):

Crash types are represented by a discrete class label from five categories: (0) none, (1) ego-only, (2) ego/vehicle, (3) vehicle-only, and (4) vehicle/vehicle. These indicate which agents are involved in the crash, with, for example, "vehicle/vehicle" describing a crash between two non-ego agents. The crash type index is embedded and projected into the cross-attention layers of the ControlNet encoder, allowing the model to generate outcomes consistent with high-level semantic intent. For clarity, examples include: (0) none — a normal driving scene with no collision; (1) ego-only — the ego vehicle veers off the road into a barrier; (2) ego/vehicle — the ego vehicle collides head-on with another car; (3) vehicle-only — a nearby car swerves and crashes into a guardrail without involving the ego vehicle; and (4) vehicle/vehicle — two other vehicles collide at an intersection while the ego vehicle observes from a distance.

6.4 Training Strategy and Setup

Figure 6.3 provides an overview of the Ctrl-Crash architecture. The system builds on the Stable Video Diffusion (SVD) framework and incorporates additional modules to enable fine-grained control over video generation via spatial and semantic conditioning. The generation process begins with two types of inputs: the initial RGB frame of a crash or driving sequence,

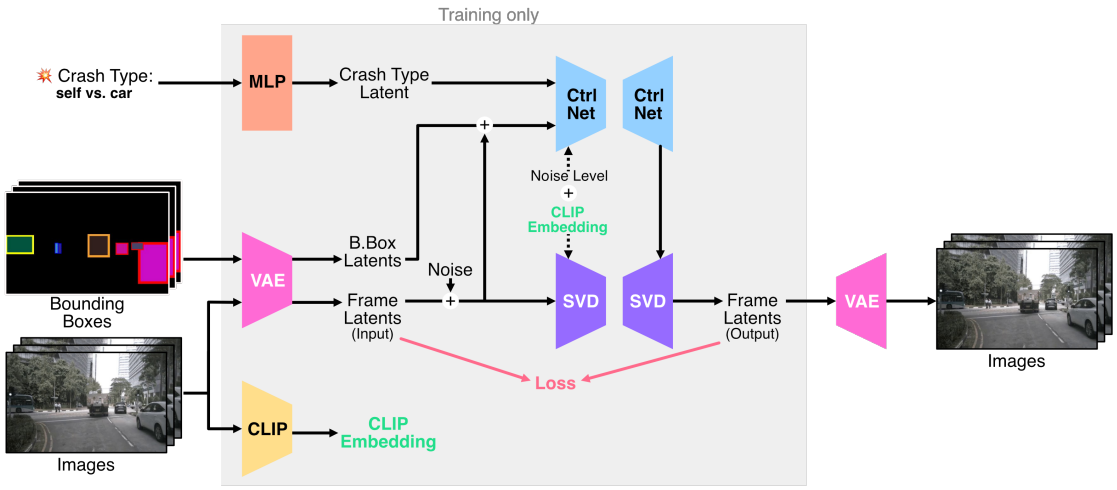


Figure 6.3 **Ctrl-Crash model architecture**: Ctrl-Crash treats Bounding Boxes (BBs) as images. Both BBs and images frames are put through a VAE image encoder. The crash type and BB embeddings are fed to ControlNet. The images embedding after adding noise (x_t), the noise level (t), and the ControlNet intermediate outputs (c) are fed to the Stable Video Diffusion (SVD) model to obtain the predicted noise $\epsilon_\theta(x_t, t, c)$. CLIP embeddings are computed by passing the first image for each video through a pretrained CLIP encoder. These CLIP embeddings are then given to the ControlNet and SVD models. on The diffusion process is solved over multiple steps using Classifier-Free Guidance in the latent space, and then decoded back to images using the VAE image decoder.

and a sequence of rendered bounding box frames that encode the motion and identity of agents over time. These bounding box sequences are represented visually and passed through a VAE encoder to obtain a latent representation. The initial frame is also encoded using the same VAE, producing frame latents that serve as the starting point of the video diffusion process. In parallel, a semantic crash type label is embedded using a simple MLP into a latent vector. This crash type embedding, along with the latent representation of the bounding box sequence, forms the conditional input to the ControlNet module. ControlNet operates as a trainable adapter to inject control signals into the base SVD model. It receives the noise level (diffusion timestep), CLIP embedding, and these structured conditioning inputs, and returns intermediate outputs that modulate the denoising process. To further enhance semantic alignment, CLIP embeddings are computed from the first image of the sequence using a pretrained CLIP encoder. These embeddings are provided both to the ControlNet and to the SVD model to help maintain consistency with the initial frame's visual and conceptual content. Table 6.2 provides an overview of the architecture's modules along with parameter counts for each.

The video diffusion process operates entirely in latent space. Gaussian noise is added to

Submodule	Status (Stage 1)	Status (Stage 2)	Number of Parameters
VAE-Encoder	Frozen	Frozen	34,163,592
VAE-Decoder	Frozen	Frozen	63,579,183
CLIP-Image Encoder	Frozen	Frozen	632,076,800
UNet	Trainable	Frozen	1,524,623,082
ControlNet	N/A	Trainable	681,221,585
Total			2,935,664,242 $\approx$ **3B**

Table 6.2 **Ctrl-Crash number of parameters per submodule**. Refer to architecture diagram in Figure 6.3 for more information on the submodules. Stage 1 and Stage 2 refer to the two stages of training for our method.

the VAE-encoded frame latents to obtain a noisy version x_t, which is then passed through the SVD UNet together with the noise level t and ControlNet outputs c. The SVD model predicts the noise $\epsilon_\theta(x_t, t, c)$ that is then used to iteratively refine the latent representation over multiple steps. Once the denoising trajectory completes, the final latent sequence is decoded back into RGB video frames using the shared VAE decoder.

The training procedure for Ctrl-Crash is organized into two sequential stages designed to progressively develop the model's generative capacity and controllability. In the first stage, the base Stable Video Diffusion (SVD) model is fine-tuned on a curated version of the MM-AU dataset (see Section 6.6). This stage follows an image-to-video generation setup, where the model is trained to reconstruct future frames from a single input image using an MSE loss in latent space:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{x_0,t,c,\epsilon\sim\mathcal{N}(0,1)}\Big[\|\epsilon - \epsilon_\theta(x_t, t, c))\|_2^2\Big], \tag{6.1}$$

where c is the image conditioning, $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon$, $\epsilon \sim \mathcal{N}(0,1)$, and $\bar{\alpha}_t$ at $t \in [1, T]$ controls the diffusion schedule. Here, ϵ_θ is the "denoiser", the core of the diffusion model, which predicts the noise added to a given input x_t at step t. See Background section 2.2 for more details on the training objective. The goal for this training phase is to adapt the pretrained SVD to the domain of ego-view driving scenes, including rare crash cases, without introducing any explicit control mechanism.

In the second stage, the fine-tuned SVD model from the first stage is frozen, and a ControlNet adapter module is trained to inject controllability into the generation process. This module enables conditioning on spatial and semantic inputs—specifically, bounding box sequences and crash type identifiers. The ControlNet integrates these signals via additional encoder

and cross-attention layers, which are appended to the frozen base UNet. The exact setup for a ControlNet adapter on a Unet model is described in Background section 2.3. All control inputs are rendered as RGB images and passed through the same pretrained VAE encoder used by SVD, maintaining a consistent representation space throughout the pipeline. This modular design allows fine-grained control over both motion and intent, while preserving the visual realism achieved during the first training stage. The training objective for the ControlNet is similar to the one from the first stage as this model (Equation 6.1) as it is a copy of the Unet model, but instead with conditioning c being the bounding box frames instead of an image frame.

To improve generalization and support controllability at inference time, randomized masking of conditioning signals is applied during ControlNet training. For bounding box inputs, a temporal dropout strategy is used: at each training step, a video timestep $k \in [0, T]$ where $T = 25$ (video length) is selected uniformly, and all bounding box frames from time k onward are replaced by a *learnable null embedding*. This embedding is distinct from a zero tensor, instead its values are optimized during the training process, ensuring that the model does not misinterpret masked information as the absence of agents. If we masked using a blank frame, this would be indistinguishable from a scene with no immediately visible agents, we therefore want to avoid confusing the model. A curriculum is employed to ease learning: bounding box masking is applied with 50% probability for the first 21k steps and then increased to 100% for the remaining 10k steps. For clarity, a 100% masking rate does not mean all bounding box frames are masked, but rather a certain random portion of them will be masked at each training step. This process aims to allow early reliance on dense control before learning to handle sparse or partial trajectories.

For semantic conditioning signals—namely, the crash type and the initial image frame—dropout is applied independently with a 10% probability for each, and a 10% chance of masking both simultaneously. This discourages overreliance on any single control modality and improves the model's robustness under varying inference-time configurations. Importantly, these dropout mechanisms enable reliable classifier-free guidance, where the model can interpolate between unconditional and conditional generation based on user-defined guidance weights.

Training is conducted using the AdamW optimizer with a learning rate of 4×10^{-5} and a batch size of 1. The first-stage fine-tuning of the base model is run for 101k steps, and the ControlNet training for 31k steps. Mixed precision is employed to reduce memory usage, where trainable modules remain in float32 while frozen components (e.g., the VAE encoder/decoder and CLIP encoder) operate in float16. The full training process was performed on 4 NVIDIA A100 80GB GPUs over approximately two weeks.

The dataset used for training comprises 7,500 crash or near-crash video clips filtered from the MM-AU dataset, with an additional 900 clips held out for validation. All evaluations reported in the Results section 6.7 are conducted using this held-out validation set.

6.5 Inference with Multi-Condition Classifier-Free Guidance

Our model supports three conditioning modalities: the initial image c_I, the bounding box frames c_B, and the crash type c_T. To enable independent control over the influence of each signal during inference, we adapt the CFG from Equation 2.24 (as described in Background section 2.3) to a multi-condition formulation, following formulations inspired by [53] and [54].

In standard CFG, both conditional and unconditional noise predictions are produced by a single model $\epsilon_\theta(\mathbf{x}_t, c)$ and $\epsilon_\theta(\mathbf{x}_t, \emptyset)$, respectively. However, as noted by recent work [55], this setup can lead to poor unconditional priors and hinder conditional fidelity. To address this, we use two separate networks: the original pretrained base model ϵ_ϕ to compute the unconditional predictions, and the fine-tuned Ctrl-Crash model ϵ_θ to compute the conditional ones. We define our multi-condition CFG formulation as follows:

$$\begin{aligned}\hat{\epsilon}_{\theta,\phi}(\mathbf{x}_t, c_I, c_B, c_T) = \epsilon_\phi(\mathbf{x}_t, c_I, \emptyset, \emptyset) \\ + \gamma_B \left[\epsilon_\theta(\mathbf{x}_t, c_I, c_B, \emptyset) - \epsilon_\phi(\mathbf{x}_t, c_I, \emptyset, \emptyset)\right] \\ + \gamma_T \left[\epsilon_\theta(\mathbf{x}_t, c_I, c_B, c_T) - \epsilon_\theta(\mathbf{x}_t, c_I, c_B, \emptyset)\right] \end{aligned} \tag{6.2}$$

Here, the first term $\epsilon_\phi(\mathbf{x}_t, c_I, \emptyset, \emptyset)$ represents the unconditional noise prediction, grounded only on the initial image c_I. We use the pretrained off-the-shelf SVD model for this term, rather than computing a fully unconditional prediction (i.e., with c_I removed), as the latter incurs substantial inference cost with minimal performance gain. The remaining terms use the fine-tuned model ϵ_θ to compute conditional noise estimates, with scalar coefficients γ_B and γ_T controlling the relative guidance strengths for the bounding box and crash type conditions, respectively. This factorized formulation enables independent modulation of each conditioning channel at inference, offering fine-grained, interpretable control over spatial and semantic properties. For instance, assigning a higher γ_T and lower γ_B prioritizes adherence to the crash type signal over strict motion patterns from the bounding boxes. In practice, we use $\gamma_B \in [1, 3]$ for bounding box control and $\gamma_T \in [6, 12]$ for crash type conditioning, with both values increasing linearly throughout the denoising process (e.g., starting at the lower bound and reaching the upper bound by the final step). All videos are sampled at 512×320 resolution for 25 frames using DDIM sampling with 30 denoising steps.

6.6 Data Preparation

Processing and preparing crash video data is a critical component of our approach, as the performance of generative models is strongly tied to the quality of their training data. In particular, crash scenarios sourced from diverse online dashcam footage—such as videos collected from platforms like YouTube—introduce unique challenges, including motion blur from sudden vehicle movements, heavy occlusion from surrounding traffic or debris, extreme lighting variations due to differing weather and time-of-day conditions, and severe compression artifacts resulting from platform-specific encoding and re-uploading. Left unaddressed, these issues can degrade the visual fidelity and temporal coherence of generated outputs. To mitigate this, we design a data processing pipeline that systematically filters low-quality samples, ensuring that the training set is both diverse and visually consistent. This careful curation not only improves the stability of training but also enhances the realism and controllability of the generated accident videos.

6.6.1 Video Processing

To train our model, we use the MM-AU dataset [51], a large-scale collection of in-the-wild dashcam crash videos sourced from public platforms. However, due to the variable quality and content of these videos, we curate the dataset using a multi-stage filtering and preprocessing pipeline. The process can be summarized as follows:

1. Filtering out low-quality, compressed, or blocky videos using a low-resolution estimation heuristic.
2. Removing clips with shot changes or unnatural scene cuts using scene change detection.
3. Normalizing the format: frame rate, resolution, and clip length.
4. Excluding scenes involving visible humans (e.g., pedestrians, cyclists, motorbikes).

In the first step, we apply frequency-domain heuristics to identify and remove videos that suffer from excessive compression artifacts or poor resolution. These issues can hinder the model's ability to learn coherent motion and object dynamics. We use a Fast Fourier Transform (FFT)-based method to detect videos with large blocky regions or low-frequency dominance—signals of strong compression or blur. This approach, described in Section 6.6.1, helps prioritize clips with high visual clarity and well-defined agent motion.

Next, we use PySceneDetect [56] to identify and discard videos containing abrupt scene transitions or camera disruptions, such as sudden viewpoint changes or dashcams that fall mid-

recording. Removing such discontinuities ensures more stable temporal consistency within each training clip.

We then normalize all video clips to a consistent resolution of 512×320 (width $\times$ height) and sample them at 6 frames per second. This not only aligns with the spatial and temporal input requirements of our model but also helps crop out unwanted overlays such as watermarks that often appear near video borders. We segment each video into fixed-length clips of 25 frames (approximately 4 seconds), discarding any samples shorter than this threshold.

Finally, for ethical and safety considerations, we exclude all videos depicting visible humans involved in crashes. This includes scenes involving pedestrians, cyclists, or motorcyclists. We manually exclude scenes where visible humans are hit to avoid exposing the model to violent content and to prevent it from learning to depict human injury, reducing the risk of harmful or inappropriate generations post-deployment.

In addition to the automated filtering steps, we also perform manual review of many video samples using internal tooling. This step helps identify edge cases and poor-quality examples that may have bypassed the automated pipeline, ensuring a cleaner and more appropriate training set. All tools and processing steps mentioned above are made publicly available on the our GitHub project page.

Low Resolution Filtering Heuristic

Algorithm 3 Estimate Upscaling Factor from Image

procedure EstimateUpsizingFactor(`image_path`)
  Load image in grayscale: $I \leftarrow$ `cv2.imread(image_path, GRAYSCALE)`
  Compute 2D FFT and shift: $F \leftarrow$ `np.fft.fftshift(np.fft.fft2`(I)`)`
  Compute magnitude spectrum: $M \leftarrow \|F\|$
  Get image size: $(h, w) \leftarrow$ `shape`(I)
  Center: $(c_x, c_y) \leftarrow (w//2, h//2)$
  Define low-frequency radius: $r \leftarrow \min(c_x, c_y)//4$
  Create circular mask L of radius r centered at (c_x, c_y)
  Compute high-frequency mask: $H \leftarrow 1 - L$
  Compute high-frequency energy: $E_{\text{high}} \leftarrow \sum M \cdot H$
  Normalize energy: $e \leftarrow E_{\text{high}}/(h \cdot w)$
  Compute upscaling factor: $f \leftarrow 1/(1 + e)$
  return f
end procedure

A key challenge in curating video datasets from online sources is ensuring that the selected samples reflect genuine high-resolution content rather than upscaled or heavily compressed

footage (via resizing). As part of the video preprocessing pipeline, we introduce a frequency-based filtering heuristic to estimate the degree of upscaling in video frames.

The heuristic is based on the observation that upscaled or low-quality images tend to exhibit diminished high-frequency content due to interpolation artifacts and smoothing. By analyzing the energy distribution in the frequency domain, we define an *upsizing factor* that acts as a proxy for the likelihood of artificial upscaling. The method is presented in Algorithm 3 and is describe in words here:

1. **Image Loading.** Each video frame is converted to grayscale to simplify analysis and focus on structural image content rather than color channels.
2. **Fourier Transform.** A 2D Fast Fourier Transform (FFT) is applied to the grayscale image to obtain its frequency representation. The spectrum is shifted so that low-frequency components are centered.
3. **Magnitude Spectrum.** The magnitude of the frequency components is computed by taking the absolute value of the FFT coefficients. This magnitude spectrum represents the intensity of different spatial frequencies in the image.
4. **High-Frequency Energy Calculation.** A circular mask is applied to exclude the low-frequency region in the center of the spectrum. The sum of magnitudes outside this central region defines the high-frequency energy.
5. **Normalization.** The high-frequency energy is normalized by the total number of pixels to produce a scale-invariant energy score.
6. **Upsizing Factor Estimation.** The final upsizing factor U is computed as: $U = \frac{1}{1+E}$, where E is the normalized high-frequency energy. This formulation ensures that frames with low high-frequency energy—indicative of potential upscaling—yield lower scores.
7. **Interpretation.** The resulting upsizing factor provides a heuristic score: lower values indicate a higher likelihood of artificial enlargement or blur, while higher values suggest more genuine, detailed high-resolution frames.

This frequency-domain metric serves as a lightweight yet effective automated quality check, helping to filter out training samples that lack meaningful visual detail. Using this metric, we automatically filtered out the 2,000 lowest scores from the dataset as these tended to appear to be very low-quality samples. This metric was also useful to guide manually filtering by hinting to which samples may be lower quality. However, manual inspection still proved to be useful as this metric is not perfect, indeed, as it is based detecting videos with little high-frequency information some videos were could wrongfully score low such as scenes containing a lot of snow, fog or darkness.

6.6.2 Datasets

Crash Type	DADA2000 Crash Types	# of Training Samples
0 - no crash	N/A	1745
1 - ego-only	13, 14, 15, 16, 17, 18, 61, 62	267
2 - ego/vehicle	1-12	3182
3 - vehicle-only	19-37, 39, 41, 42, 44	577
4 - vehicle/vehicle	38, 40, 43, 45-51	2168

Table 6.3 **Ctrl-Crash to DADA2000 Crash Type association**. We also report the number of samples for each type used for training Ctrl-Crash. See Appendix C for a visual representation of all DADA2000 crash types.

Following the filtering steps described in Section 6.6.1, we retain approximately 7,500 videos from the original 11,727 provided in the MM-AU dataset. These filtered videos are split into a training set and a held-out test set using stratified sampling across accident type categories, maintaining a 90/10 ratio to preserve class balance.

All annotated video samples—spanning our MM-AU extension, the RussiaCrash test set, and other sources—are stored in structured JSON format, where each annotation file corresponds to a single video. Each annotation contains three key components: the `video_source` field identifies the filename of the original video, the `metadata` field encodes scenario-level information such as whether the ego vehicle is involved in the accident and the categorical accident type index as defined in the DADA2000 dataset [57], and the `data` field contains a per-frame list of object annotations. Each frame entry includes the frame filename and a list of labeled objects, each specified by a persistent `track_id`, a semantic class label (e.g., "car" or "person"), a numerical class index (e.g., 0 = person, 1 = car), and a bounding box defined as normalized coordinates in the format $[x_{min}, y_{min}, x_{max}, y_{max}]$. All objects are temporally linked using consistent `track_id` values, and class labels follow the taxonomy of the YOLOv8 model used during annotation. A summary of the exact annotation structure is provided in Appendix D.

To simplify semantic control during training, we reduce the original set of accident types provided in MM-AU (defined by DADA2000) to a coarser set of five categories, as described in Section 6.3.3: (0) no crash, (1) ego-only crash, (2) ego/vehicle, (3) vehicle-only, and (4) vehicle/vehicle. The mapping from the original DADA2000 categories to this reduced taxonomy is provided in the first two columns of Table 6.3.

For training and evaluation, each video is segmented into 25-frame clips. We use the per-frame labels provided by MM-AU to identify when the accident occurs and to extract sequences of "abnormal" and "normal" driving behavior. For each video, we sample one 25-frame clip

containing the accident and label it using the corresponding reduced crash type. If available, we also sample one 25-frame clip that consists entirely of normal driving behavior—i.e., frames not overlapping with any labeled abnormal or crash intervals—and assign it the label "no crash". This procedure yields a total of 6,927 clips containing crash events and 1,964 clips of normal driving. Class-specific statistics are shown in Table 6.3.

6.6.3 YOLO-SAM: Hybrid Bounding Box Annotation Pipeline

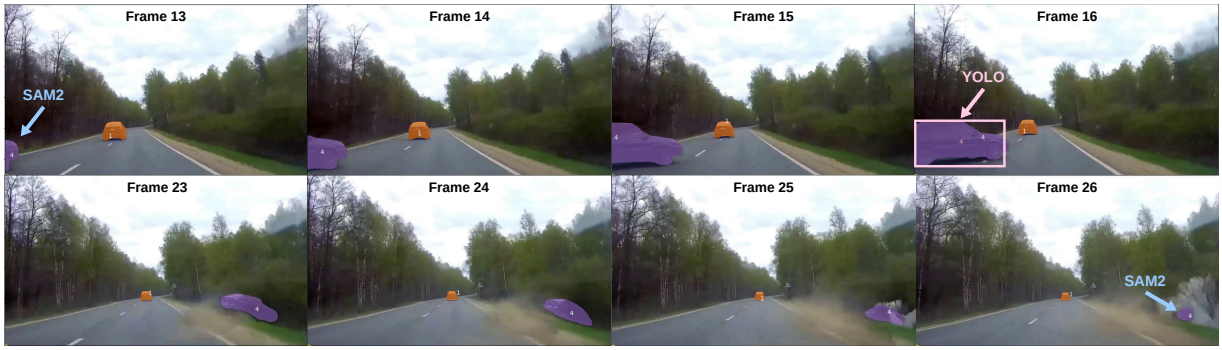


Figure 6.4 **Example of bounding box tracking for a vehicle veering off the road**. SAM2 segmentations for the two vehicles are shown in purple and orange. YOLOv8 first detects the purple car only starting at frame 16 (light pink box), while SAM2 successfully infers its presence across many more frames by propagating the segmentation bidirectionally in time. This provides more complete and temporally consistent annotations, especially in cases where objects become partially occluded or deformed.

Accurately annotating bounding boxes in dashcam crash footage is a challenging task, especially due to the unpredictable visual conditions often present in such scenes. Occlusions, motion blur, partial visibility, and sudden object deformation can easily confuse conventional object detectors and trackers. This is particularly problematic in high-stakes scenarios like vehicle collisions, where precise temporal tracking of objects is crucial for training generative models that aim to simulate or predict realistic crash behavior.

We develop a hybrid annotation pipeline, YOLO-SAM, to generate high-quality bounding boxes for visible road users in crash and driving videos. This combines the speed and class-awareness of YOLOv8 [58] with the instance-level segmentation and temporal propagation capabilities of SAM2 [59]. The hybrid design addresses detection failures due to occlusion, deformation, or sudden motion—common in crash scenarios—and supports dynamic entry/exit of agents.

Figure 6.4 illustrates a motivating example where a vehicle begins to leave the road and gradually disappears into a ditch. While YOLO detects the vehicle late after it enters the

frame and loses it long before it falls into the ditch,(i.e., only when it is clearly visible and identifiable), SAM2 is able to infer its presence over a much longer temporal window by propagating its segmentation mask both backward and forward in time. This enables the recovery of early and late-frame annotations that YOLO misses entirely. Such temporal reasoning is essential in crash scenarios, where agents often undergo rapid visual changes or momentarily exit the scene due to impact or occlusion.

YOLOv8 is run frame-by-frame to detect standard object classes (e.g., cars, trucks, buses) and assign initial bounding boxes and track IDs. While fast and accurate under normal conditions, YOLO has several shortcomings:

1. **Duplicate Detections:** YOLO can assign multiple detections to the same object over time. We reject predictions if their intersection over union (IoU) with previous detections exceeds 0.8.
2. **Tracking Loss:** YOLO may lose an object (e.g., under occlusion) and redetect it with a new track ID. These failures are corrected using SAM2.

To address YOLO's limitations, we use SAM2 to refine and extend YOLO detections:

1. **Shape Correction:** SAM2 provides temporally consistent masks, reducing YOLO's tendency to distort box sizes and shapes.
2. **Redetection Verification:** When YOLO redetects a lost object with a new ID, we compare its spatial overlap with SAM2's mask to either restore the original ID or assign a new one.
3. **Track ID Switches:** If YOLO mistakenly assigns an existing ID to a new object, SAM2's predictions are used to detect the mismatch and correct the ID.
4. **Early-frame Recovery:** SAM2 supports bidirectional propagation, allowing recovery of early frames missed by YOLO. However, to avoid hallucinated presence, we only accept early-frame completions up to a few frames before YOLO's first detection, and only when confidence is high.

For each YOLO-detected object, SAM2 is prompted to generate per-frame instance masks. These masks are converted to tight bounding boxes, and tracking continuity is enforced by ID consistency checks based on IoU overlap. The result is a per-frame annotation of bounding boxes with consistent track IDs and object classes.

Compared to YOLO or SAM2 alone, the hybrid approach provides significantly more complete and robust annotations, crucial for training controllable video generation models like Ctrl-Crash. The full implementation is available in our project repository for reproducibility.

6.7 Quantitative Evaluation

We evaluate the generation quality of Ctrl-Crash across three core settings: (1) general crash video generation in Table 6.4; (2) varying the number of bounding box guidance frames provided in the prediction task to assess the impact of partial trajectory information in Table 6.5; and (3) varying the crash type control signal to assess counterfactual generation in Table 6.6. All generated videos are 25 frames long at a resolution of 512×320, and metrics are computed over 200 video samples unless otherwise specified.

For quantitative evaluation, we compare Ctrl-Crash against several recent methods for driving or crash video generation. AVD2 [60] and OAVD [51] focus on generating videos of vehicle accidents, but their primary goal lies in accident description and reasoning rather than high-fidelity, controllable crash synthesis. DrivingGen [61] generates crash scenarios from textual accident reports, translating structured natural language into plausible video depictions of the described events. We also compare to Ctrl-V [41], introduced in Chapter 4, which achieves high visual fidelity but was not designed to produce realistic crash events. As an additional baseline, we include the SVD Base model, the pretrained Stable Video Diffusion model without fine-tuning for crash generation, to assess the gains provided by crash-specific adaptation and conditioning.

6.7.1 Evaluation Metrics

We evaluate the quality of generated videos using both distributional video-level metrics and frame-level fidelity metrics. FVD and JEDi are video-level metrics that measure the distance between the distributions of generated and real videos (lower is better). JEDi addresses limitations of FVD by relaxing the Gaussian assumption and tend to converge with fewer samples. Together, these metrics provide complementary views of video realism at the distributional level. These metrics are detailed in Section 2.4.1 of the background chapter.

To evaluate distributional similarity between generated and real videos, we compute FVD and JEDi under two complementary evaluation protocols for the different methods that will be presented in the following section, which differ based on how the ground-truth distribution is constructed:

1. **Condition-Aligned Evaluation:** For models such as Ctrl-Crash, Ctrl-V, and SVD Base, we have access to the ground-truth data used to seed generation—namely, the initial frame, bounding box trajectories, and even crash type category. In this evaluation protocol, we sample 200 videos from our held-out validation set, and generate

corresponding videos using their conditioning information. We do not directly compare generated videos to their exact ground-truth counterpart; instead, we treat these 200 ground-truth videos as a reference distribution that is semantically aligned with the generated one (e.g., similar vehicle types, scene layouts, and crash categories). FVD and JEDi are then computed between these two condition-aligned distributions. This setup ensures that the real and generated samples share similar structural priors, making the distributional comparison more meaningful. This method is used for the FVD and JEDi results reported in Table 6.5 and Table 6.6.

2. **Random GT Evaluation:** Some baselines, such as AVD2, only provide generated videos without access to the original conditioning inputs, preventing us from constructing a semantically aligned ground-truth distribution. To ensure fair comparison in such cases, we instead sample 500 videos randomly from the MM-AU validation set to represent the ground-truth distribution. We then compare this to 200 generated samples from each model (including AVD2, Ctrl-Crash, Ctrl-V, and SVD Base). While this evaluation provides less precise alignment between real and generated content, it allows for inclusion of baselines where seed information is unavailable. This method is used for FVD and JEDi results reported in Table 6.4.

By using both evaluation protocols, we balance semantic alignment (when possible) with broad comparability, enabling fair and informative assessments across different classes of models.

In addition, we report frame-level metrics that directly assess perceptual quality relative to ground truth frames. These include LPIPS, SSIM, and PSNR. These metrics quantify visual similarity, structural consistency, and reconstruction fidelity, respectively, offering finer-grained insight into how well the generated frames preserve appearance and local coherence. To extend these frame-level metrics to video, we average the scores for all the frames of a given video. See Section 2.4.2 of the Background for more detailed information on the implementation of these metrics.

6.7.2 Results

In Table 6.4, we report video quality metrics across different video diffusion methods and baselines. For methods with available video samples, we compute the FVD and JEDi video metrics. For OAVD and DrivingGen, we report the FVD values cited in their original publications due to the unavailability of code or generated samples. These results are marked with an asterisk (“*”) in the table, to indicate that they might not be directly comparable due

Method	Conditions	FVD↓	JEDi↓	LPIPS↓	SSIM↑	PSNR↑
OAVD	img + BB + text	5238*	-	-	-	-
DrivingGen	img + BB + text	978.0*	-	-	-	-
SVD base	img	1420	3.628	0.5800	0.3074	11.74
AVD2	img + text	1321	2.029	-	-	-
Ctrl-V	img + BB	517.1	0.2910	0.3670	0.5372	16.81
Ctrl-Crash	img + BB + action	**449.5**	**0.1219**	**0.3113**	**0.5836**	**18.33**

Table 6.4 **Comparison of accident video generation quality across diffusion-based methods.** We report FVD, JEDi, and LPIPS scores (↓ lower is better) along with SSIM and PSNR scores (↑ higher is better). Scores marked with * are taken directly from the original papers and may not be strictly comparable due to differences in evaluation setup. The "Conditions" column describes inputs used as control signals. "img": initial image frame, "BB": bounding box frames, "text": textual description, "action": discrete crash category value. We substitute a dash ("-") for results that couldn't be computed, due to restricted access to data or models

to possibly different evaluation protocols. Frame-level metrics (LPIPS, SSIM, and PSNR) are computed for methods where we had access to the GT videos used to seed the generated videos.

From the results in the table, we observe that Ctrl-Crash achieves the best results across all metrics among compared methods, indicating strong alignment with real crash dynamics and superior video quality. The FVD and JEDi metrics were computed using the "Conditioned-Aligned Evaluation" method defined previously. The SVD Base model, which serves as the foundation for Ctrl-Crash, performs poorly, as it was not trained for driving or crash-related content. Ctrl-V, while similar in architecture, lacks crash-specific training data and semantic control, leading to notable quality degradation near the crash event. AVD2 performs poorly and exhibits inconsistent visual quality and weaker temporal coherence compared to Ctrl-Crash, as confirmed by FVD and JEDi. These results highlight the importance of both targeted training and structured control for crash simulation. LPIPS, SSIM, and PSNR are also consistent with the video-level metrics, suggesting improved crash video quality that is closer to the GT videos.

We also study the impact of varying the number of bounding box frames used as conditioning for Ctrl-Crash, in Table 6.5. Specifically, we compare using partial bounding-box frame conditioning from zero frames to a few bounding box frames (*Crash Prediction* task) all the way to a fully defined motion sequence by providing all 25 frames (*Crash Reconstruction*). As shown in the table, generation quality improves consistently with the number of provided

#BBs	FVD↓	JEDi↓	LPIPS↓	SSIM↑	PSNR↑
0 (none)	422.1	0.3155	0.3856	0.5188	16.57
3	375.7	0.2949	0.3594	0.5434	17.27
9	353.3	0.2160	0.3392	0.5614	17.83
25 (all)	**323.9**	**0.1219**	**0.3113**	**0.5836**	**18.33**

Table 6.5 **Effect of bounding box conditioning on Ctrl-Crash crash video prediction quality.** We evaluate Ctrl-Crash on the *Crash Prediction* task by varying the number of initial bounding box frames provided as input (out of 25 total frames).

bounding box frames. This trend is visible across both distributional metrics (FVD and JEDi) and frame-level scores (LPIPS, SSIM, PSNR), and supports the hypothesis that denser spatial constraints lead to easier prediction tasks and more stable outputs. The results support that Ctrl-Crash gracefully interpolates between unconditional prediction and fully supervised reconstruction.

Crash Type	FVD↓	JEDi↓
GT crash type	375.7	0.2949
0 - no crash	400.9	0.4514
1 - ego-only	379.5	0.3091
2 - ego/vehicle	372.9	0.3001
3 - vehicle-only	398.1	0.3856
4 - vehicle/vehicle	383.4	0.3416

Table 6.6 **Effect of crash type conditioning on crash video generation quality.** We evaluate Ctrl-Crash on the *Crash Counterfactuals* task by varying the crash type conditioning and nothing else. For each case, 200 videos were generated using the same initial image and three bounding box frames as initial context, but with different desired crash types.

Finally, in Table 6.6, we evaluate Ctrl-Crash on the *Crash Counterfactuals* task, where we vary only the crash type conditioning while keeping all other inputs fixed. Specifically, we select 200 ground-truth videos and generate five counterfactual versions for each, using the same initial GT image and the first three bounding box frames, but conditioning on different desired crash types. In other words, starting from the same initial scene, we explore five different possible outcomes: (0) no crash, (1) ego-only crash, (2) ego and other vehicle crash, (3) other vehicle only crash, and (4) multi-vehicle crash.

For each crash type, we generate 200 videos and compute FVD and JEDi to assess video quality. The lowest (i.e., best) scores are observed for the ego/vehicle (type 2) crash condition. Referring to Table 6.3, this is also the most frequent crash type in the training set, suggesting

that the model may have learned to represent it more reliably. However, this trend does not hold across all categories: for instance, ego-only (type 1) crashes are the least represented in the training data, yet achieve the second-best FVD and JEDi scores. One plausible explanation is that ego-only crashes often do not involve visible impact with other agents (e.g., the ego vehicle veering off-road or falling into a ditch) making them easier to render realistically. In contrast, generating a vehicle-only crash (type 3), which requires animating an isolated accident involving another vehicle (e.g., another car crashing alone), may be more visually complex. This is supported by the higher FVD and JEDi scores observed for this category.

For reference, the first row of Table 6.6 reproduces results from Table 6.5, where the desired crash type matches the ground-truth crash type. These samples are randomly drawn across all crash categories. Overall, both FVD and JEDi scores in the counterfactual setting remain relatively close to the matched ground-truth baseline, suggesting that the model can generate plausible alternate outcomes while preserving visual fidelity. However, the consistent degradation in metric scores across most counterfactual categories highlights a key challenge: reliably deviating from the initial conditions to create realistic alternative futures. One hypothesis for this performance gap is that, given the short temporal window (25 frames, or roughly 4 seconds), it may be inherently difficult for the model to diverge substantially from the visual cues of the initial frame. For example, if the input clearly depicts two vehicles speeding toward a head-on collision, it may be unrealistic for the model to generate a "no crash" or "ego-only" crash alternative. As discussed in Section 6.5, the Ctrl-Crash framework supports adjusting the strength of each control signal via CFG. In principle, this mechanism enables giving more weight to the crash type conditioning signal when generating counterfactuals. Further work is needed to explore whether this tuning improves the model's ability to override visually strong cues in the initial input and produce high-fidelity, diverse outcomes.

Across all benchmarks, Ctrl-Crash shows significant improvements over prior diffusion-based model for crash generation. It handles both unconstrained and highly conditioned inputs, demonstrating strong performance in generating diverse crash outcomes and plausible reconstructions. In the next section, we complement these quantitative evaluations with a user study and qualitative visualizations.

6.8 Qualitative Evaluation

In this section we will complement the quantitative results presented in Section 6.7 with visual examples to illustrate the model's behaviour[1].

First, in the *crash reconstruction* task (Figure 6.5), the model is conditioned on the full set of inputs: the initial frame, the complete sequence of bounding box trajectories, and the ground-truth crash type. This setup allows us to evaluate whether Ctrl-Crash can faithfully reconstruct the original crash sequence in terms of both motion and appearance. In many cases, the generated video closely matches the ground truth—agents appear in the correct spatial configuration, the motion trajectory is plausible, and the overall visual coherence with the initial frame is preserved. In this task, the motion, position and number of visible agents is fully constrained by the provided bounding boxes, so there is minimal diversity between different seeds. Visually, this also tends to lead to high quality video samples compared to other tasks with partial input.

Second of all, we show several examples of clips generated by Ctrl-Crash according to the *crash prediction* task (Figure 6.6). We condition the model with 9 initial bounding box frames, an initial image and a desired crash type. The goal for this task is for the model to extrapolate from partial trajectory information the future motion of the agents in the scene towards the desired crash type by generating realistic frames that follow the visual style of the provided initial frame. Interestingly, when the input sequence is derived from an actual crash, and the desired crash type matches, we often observe sequences that closely resemble the original ground truth. However, this is not always the case, as the model often has a lot of freedom to generate novel agents, determine which specific agents should be involved in the crash, and to determine the exact appearance of agents that were not fully visible in the initial frame.

Third of all, in the *crash counterfactual* task (Figure 6.7), we extend the *prediction* task by varying the crash type conditioning. The goal is to generate multiple plausible future scenarios that could have emerged from the same starting point, depending on how the dynamics unfold. For each input scene, we have the possibility to generate five variations: (0) no crash, (1) ego-only crash, (2) ego/vehicle crash, (3) vehicle-only crash, and (4) vehicle/vehicle crash. This showcases the model's ability to alter narrative outcomes while preserving consistency with the initial visual and motion context.

We are otherwise not restricted to seeding the generation process with crash videos. To

[1] Animated video examples for all the use cases described in this section can be viewed on the Ctrl-Crash project website: `https://anthonygosselin.github.io/Ctrl-Crash-ProjectPage/`.

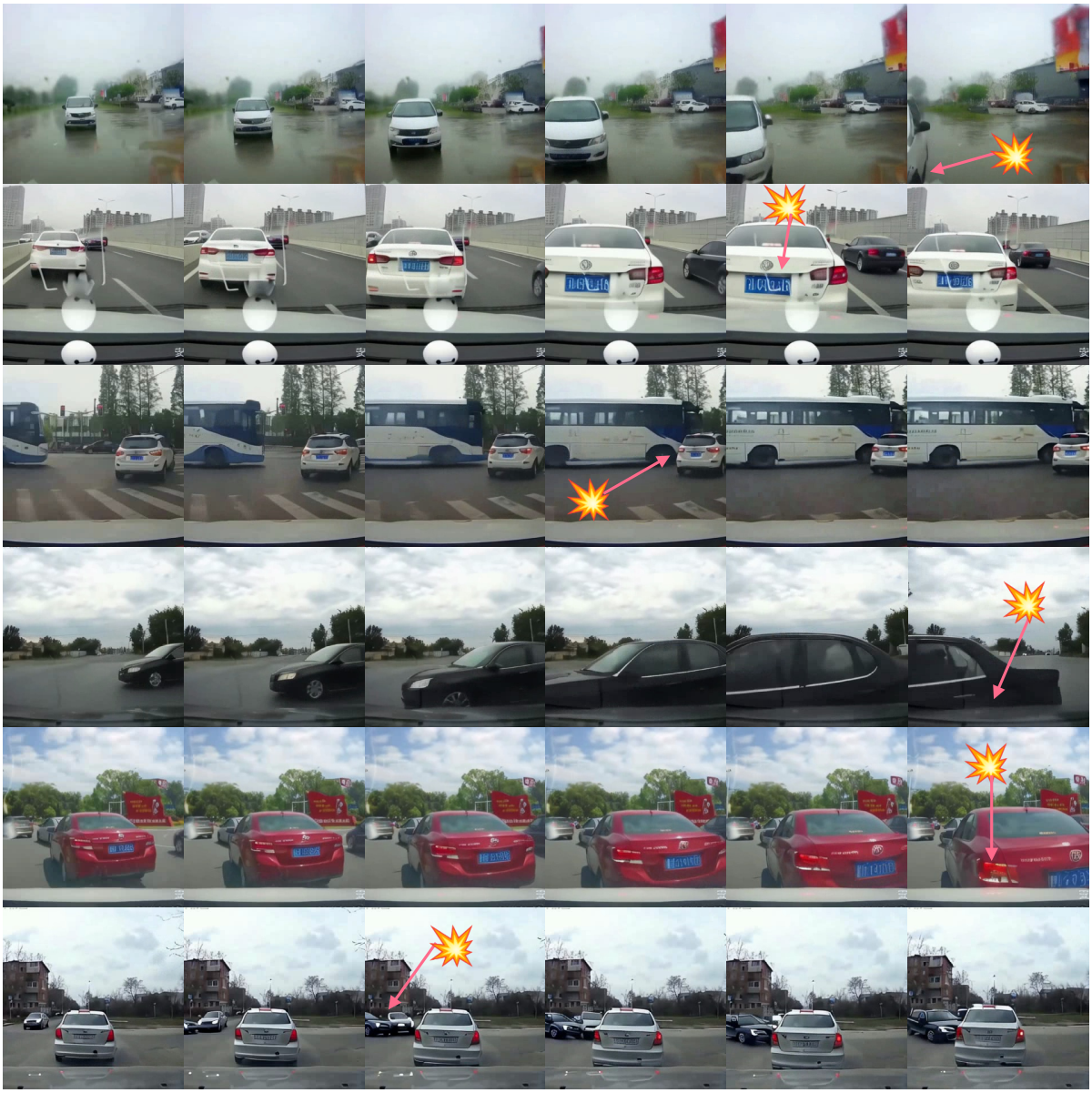


Figure 6.5 **Ctrl-Crash qualitative results conditioned on 25 (all) bounding box frames.** We indicate with a pictogram the moment and location of the crashes.

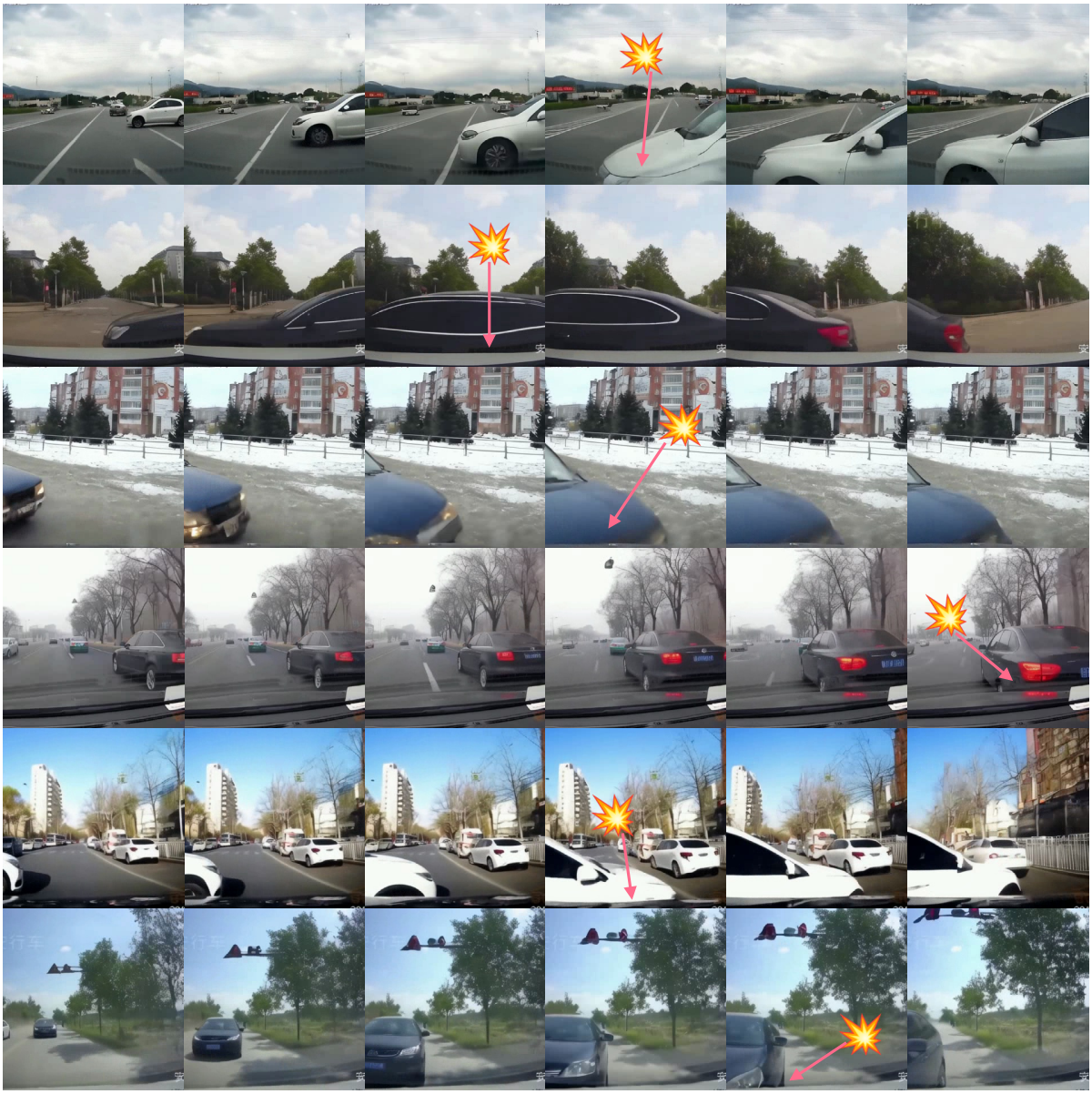


Figure 6.6 **Ctrl-Crash qualitative results conditioned on an initial 9 bounding box frames.** The first two frames of these sequences were conditioned on bounding box frames, but not the others. We indicate with a pictogram the moment and location of the crashes.

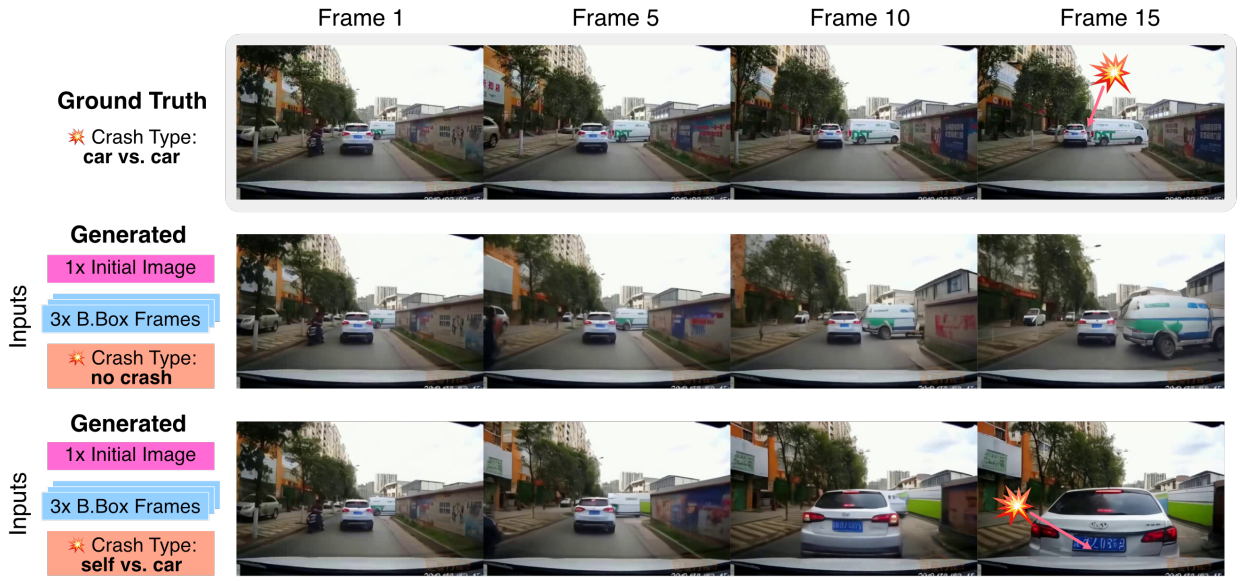


Figure 6.7 **Counterfactual Crash Generation**: this diagram demonstrates the ability of our model to generate counterfactual crashes (**Middle**: no crash, **Bottom**: ego/car crash) while beginning from the initial frame and 3 bounding-boxes frames of the real video (**Top**: the real car crash).

demonstrate the practical utility of Ctrl-Crash for safety-critical dataset augmentation, we apply our model to generate crash scenarios seeded from the BDD100k dataset, an established real-world driving dataset that contains no actual crashes. By conditioning on the inital frame, 9 initial bounding box frames, and, in these examples, the "ego/vehicle" (type 2) crash type for the BDD100k dataset, Ctrl-Crash is able to hallucinate plausible accident outcomes, transforming otherwise uneventful driving clips into diverse crash scenarios. This capability is especially valuable for enriching datasets used in autonomous driving research, where real crash data is scarce, sensitive, or ethically challenging to collect. Synthesizing rare and hazardous events from benign scenes enables safer training and evaluation of perception and planning systems without requiring exposure to real-world danger. In Figure 6.8, we show some examples of car crash videos generated by Ctrl-Crash from non-crash scenes in the BDD100k dataset.

Additionally, in Figure 6.9, we present qualitative comparisons between Ctrl-Crash and three other diffusion-based video generation methods: AVD2, DrivingGen, and Ctrl-V. AVD2 and DrivingGen represent the most relevant baselines for crash video generation, while Ctrl-V serves as a high-quality general-purpose method that, although capable of producing realistic-looking videos, lacks explicit support for crash synthesis. Visually comparing random samples across these methods, Ctrl-Crash appears to consistently generate more plausible and detailed

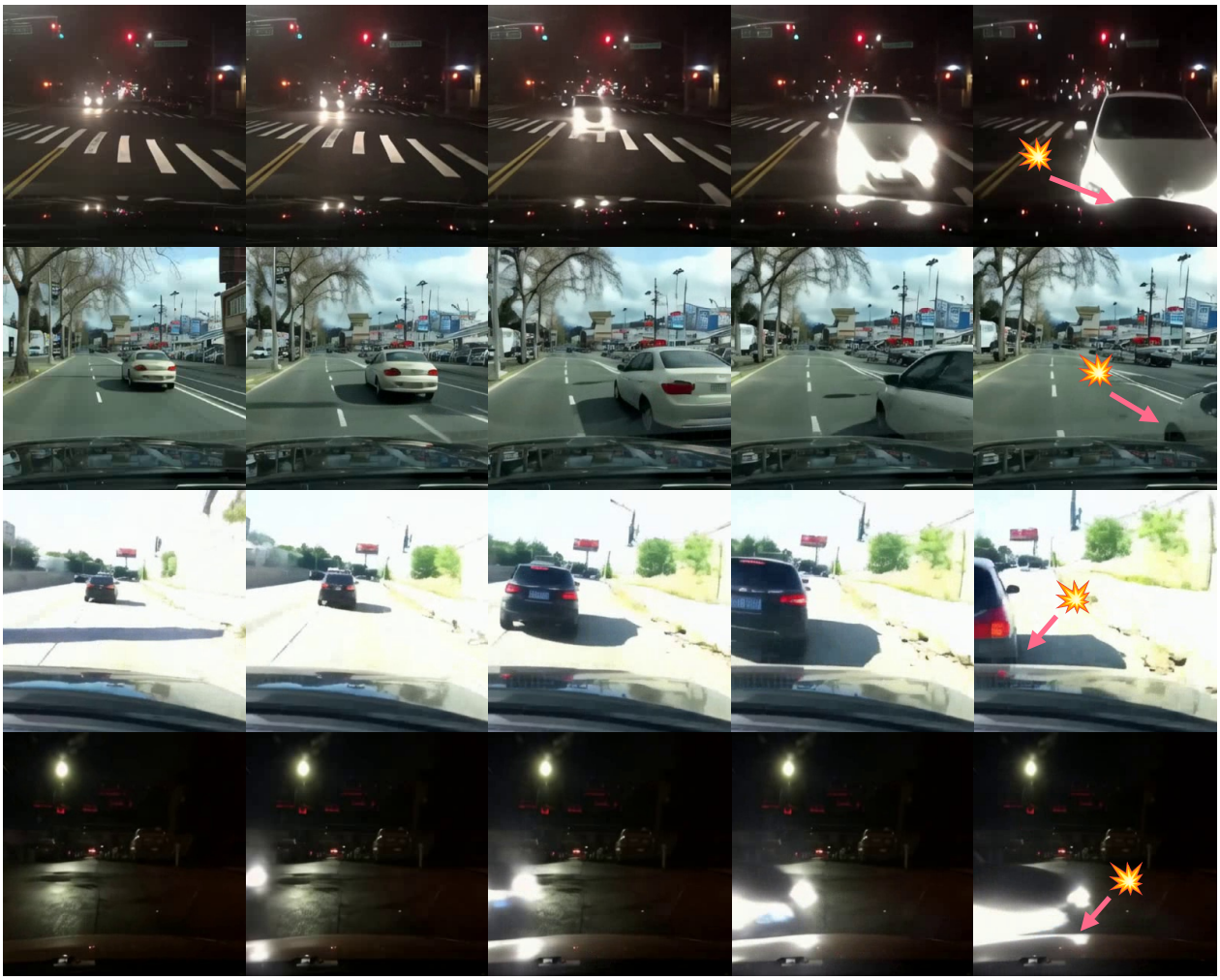


Figure 6.8 **Crash scenarios generated by Ctrl-Crash from non-crash scenes in the BDD100k dataset**. Each row shows 5 frames from a generated 25-frame clip. Despite originating from benign driving scenes, the model produces visually coherent and diverse crash outcomes across examples.

Figure 6.9 **Qualitative results comparing AVD2, DrivingGen, Ctrl-V, and Ctrl-Crash**. The crash generated by AVD2 is visually shaky, with scenes that often lack consistency. Driving-Gen also produces low-quality and choppy videos. While Ctrl-V achieves good visual quality, it fails to generate realistic crash events. In contrast, Ctrl-Crash outperforms all baselines in both visual fidelity and scene consistency, while accurately modeling crash dynamics. The "Crash?" column indicates whether a crash occurs in the video, which is often hard to view in individual frames.

crash scenarios. This observation aligns with the quantitative results reported in Section 6.7. However, we acknowledge that such visual comparisons are subject to observer bias and do not constitute a rigorous qualitative evaluation. Note that additional comparisons are shown in Appendix B.

6.9 Conclusion

In summary, this chapter presented Ctrl-Crash, a controllable video diffusion framework tailored for realistic and semantically diverse crash scenario generation. By combining spatial trajectory conditioning with high-level crash type intent, the model bridges fine-grained motion control and counterfactual reasoning, enabling the synthesis of safety-critical scenarios that are both visually realistic and behaviorally plausible. A key enabler of this

approach was the dedicated data processing pipeline, which included automated bounding box detection, tracking, and rigorous filtering of low-quality samples to ensure clean and reliable training data. Through rigorous quantitative and qualitative evaluation, Ctrl-Crash demonstrated state-of-the-art performance among current diffusion-based crash generation approaches, highlighting its potential as a scalable tool for advancing autonomous vehicle testing and safety analysis.

CHAPTER 7 CONCLUSION

Ensuring the safety and reliability of autonomous vehicles demands rigorous evaluation across a wide range of driving scenarios, including rare, dangerous, and unpredictable edge cases. Yet, the scarcity of real-world data for such events poses a significant challenge to the development and validation of robust AV systems. In this context, the ability to simulate long-tail scenarios in a controllable and visually realistic manner has emerged as a critical need in the field of autonomous driving research.

This thesis addressed this challenge by exploring data-driven generative approaches for simulating safety-critical driving scenarios, with a particular emphasis on controllable video synthesis of car crashes. The central hypothesis guiding this work was that generative models—trained on real-world footage and conditioned on interpretable signals—could provide a scalable, flexible, and expressive framework for generating rare driving events in high fidelity.

7.1 Summary of Works

In this thesis we began by examining two prior works, CtRL-Sim and Scenario Dreamer, which laid the foundation for simulating multi-agent driving behavior in bird's-eye view. CtRL-Sim introduced a controllable multi-agent Transformer model capable of steering simulated agents toward diverse reward-driven behaviors. Scenario Dreamer complemented this by learning to generate the BEV environment, including object and map structure, using a data-driven prior. Together, these works formed a pipeline for generating richly structured BEV scenarios in a controllable and semantically grounded way.

Building on these ideas, we introduced Ctrl-V, a two-stage generative pipeline that bridged the gap between BEV simulation and photorealistic ego-view video. The first stage, the Bounding Box Generator, produced interpretable multi-agent trajectories in the form of rasterized bounding box sequences. These served as the controllable input to the second stage, the Box2Video model, which synthesized first-person video using a diffusion model architecture. This formulation enabled controllable video generation from intermediate symbolic representations, supporting interpretability and semantic manipulation.

Finally, we culminated to the core contribution and goal of this thesis by presenting Ctrl-Crash, a novel generative framework for realistic and controllable crash video synthesis. Ctrl-Crash extended latent video diffusion models with conditioning mechanisms based on both spatial (bounding box) and semantic (crash type) control signals. Through a two-

stage training pipeline—first fine-tuning on crash data, then introducing conditioning via ControlNet adapters—the model was able to produce high-quality, diverse crash videos from a single initial frame. A key feature of Ctrl-Crash was its support for counterfactual generation, enabling the synthesis of alternate crash outcomes from the same initial state, and thereby offering a new tool for causal reasoning and safety analysis. Additionally, all work introduced in the main chapters of this thesis to which the author contributed to were made fully open-source to promote further research and scientific discovery.

7.2 Limitations and Future Research

While this thesis has advanced the state of controllable, realistic simulation for safety-critical autonomous driving scenarios, several limitations remain that open promising avenues for future research.

First, generating counterfactual outcomes remains challenging when the initial scene strongly suggests a different crash trajectory than the one being conditioned on. In short temporal horizons—such as the 4-second sequences used in Ctrl-Crash—it is often difficult for the model to deviate meaningfully from visually or kinematically constrained starting conditions. Second, current controllability is grounded in 2D bounding box trajectories. Although effective for specifying general agent motion, these representations omit depth, orientation, and rotational dynamics, making behaviors such as spinouts or rollovers harder to represent. Future extensions may explore 3D bounding boxes or richer trajectory representations to overcome this. The overall reliance on automatically detected bounding boxes also introduces sensitivity to detection and tracking errors, particularly in fully conditioned reconstruction settings. Third, the semantic intent space is defined by a fixed set of discrete crash type labels. While simple and effective, this formulation constrains expressiveness and prevents the model from capturing finer-grained distinctions or novel crash categories. Incorporating natural language as a conditioning signal could unlock more nuanced control and allow the model to interpret detailed scene descriptions or intentions that go beyond predefined categories and bounding-box information. Similarly, the current BEV2POV framework, while demonstrating a compelling proof of concept for translating high-level BEV simulations into photorealistic POV videos, has yet to be developed into a scalable, production-ready system that can operate in a closed-loop AV simulation pipeline. Finally, the generated crash scenarios, though visually realistic, are not physically validated. Without grounding in a physics model, some agent motions and collisions may diverge from real-world dynamics, limiting their reliability for certain simulation-based safety evaluations.

Looking forward, several research directions can address these limitations. A key prior-

ity is to fully realize the BEV2POV framework, enabling seamless, scalable translation of behavior-level simulations from CtRL-Sim and Scenario Dreamer into photorealistic ego-POV sequences. Expanding controllability through natural language descriptions, HD map context, and multimodal sensor fusion could allow more nuanced and interpretable scenario specification. Replacing 2D bounding boxes with richer 3D motion and orientation cues—and optionally integrating differentiable physics engines—would help ensure both visual fidelity and physical plausibility.

Advances in large-scale foundation video models, such as NVIDIA's Cosmos, offer another path forward. Fine-tuning these models for AV crash simulation could extend temporal horizons, improve visual quality, and capture more complex multi-agent interactions. Such models could also be leveraged to augment large-scale AV datasets with diverse, rare crash and near-crash scenarios, enhancing safety coverage where real-world data is scarce or impractical to collect.

Finally, future work should place greater emphasis on bridging the sim-to-real gap. Systematic evaluation of generated scenarios within downstream AV perception and planning stacks, combined with domain adaptation and domain randomization techniques, will be critical to ensuring that synthetic outputs transfer effectively to real-world operational contexts.

In this way, the methods developed in this thesis lay the groundwork for a new generation of data-driven, controllable, and photorealistic simulation tools that can support the safe deployment of autonomous vehicles in complex, safety-critical environments. Neural network-based simulators built on diffusion models, or other generative approaches, are rapidly transforming how we think about testing robotics and autonomous systems. It is plausible that, in the near future, the majority of simulation for vehicles and robots will be performed in learned world models that are indistinguishable from reality. Indeed, the roads of the future will first be built in code; where every crash is safe, every near-miss is a lesson, and infinite variations of the world can be generated from a simple prompt before even a single wheel touches asphalt. Until then, we will continue working toward better, more faithful simulators that narrow the gap between the virtual and the real.

REFERENCES

[1] J. Fan, "The physical turing test: Jim fan on nvidia's roadmap for ...," YouTube video, May 2025, video published approximately 3 months before August 2025. [Online]. Available: https://www.youtube.com/watch?v=_2NijXqBESI

[2] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[3] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[4] K. Zakka, B. Tabanpour, Q. Liao, M. Haiderbhai, S. Holt, J. Y. Luo, A. Allshire, E. Frey, K. Sreenath, L. A. Kahrs, C. Sferrazza, Y. Tassa, and P. Abbeel, "Mujoco playground," 2025. [Online]. Available: https://arxiv.org/abs/2502.08844

[5] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "Gpu-accelerated robotic simulation for distributed reinforcement learning," 2018. [Online]. Available: https://arxiv.org/abs/1810.05762

[6] NVIDIA Corporation, "Nvidia drive sim: End-to-end simulation for autonomous vehicle development," 2025, accessed: 2025-08-08. [Online]. Available: https://developer.nvidia.com/drive/simulation

[7] A. Blattmann, T. Dockhorn, S. Kulal, D. Mendelevitch, M. Kilian, D. Lorenz, Y. Levi, Z. English, V. Voleti, A. Letts, V. Jampani, and R. Rombach, "Stable video diffusion: Scaling latent video diffusion models to large datasets," 2023. [Online]. Available: https://arxiv.org/abs/2311.15127

[8] N. Agarwal, A. Ali, M. Bala, Y. Balaji, E. Barker, T. Cai, P. Chattopadhyay, Y. Chen, Y. Cui, Y. Ding *et al.*, "Cosmos world foundation model platform for physical ai," *arXiv preprint arXiv:2501.03575*, 2025.

[9] T. Brooks, B. Peebles, C. Holmes, W. DePue, Y. Guo, and OpenAI, "Video generation models as world simulators," https://openai.com/index/video-generation-models-as-world-simulators/, 2024, technical report.

[10] DeepMind, "Veo 3: Google's text-to-video model with native audio," https://deepmind.google/models/veo/, 2025, released May 2025; model card published May 23, 2025.

[11] A. Melnik, M. Ljubljanac, C. Lu, Q. Yan, W. Ren, and H. Ritter, "Video diffusion models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2405.03150

[12] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," *Advances in neural information processing systems*, vol. 32, 2019.

[13] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2256–2265. [Online]. Available: https://proceedings.mlr.press/v37/sohl-dickstein15.html

[14] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020. [Online]. Available: https://arxiv.org/abs/2006.11239

[15] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, "Video diffusion models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 8633–8646. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/39235c56aef13fb05a6adc95eb9d8d66-Paper-Conference.pdf

[16] Y. He, T. Yang, Y. Zhang, Y. Shan, and Q. Chen, "Latent video diffusion models for high-fidelity long video generation," 2023. [Online]. Available: https://arxiv.org/abs/2211.13221

[17] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," 2022. [Online]. Available: https://arxiv.org/abs/2010.02502

[18] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022. [Online]. Available: https://arxiv.org/abs/2207.12598

[19] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," 2023. [Online]. Available: https://arxiv.org/abs/2302.05543

[20] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," 2021. [Online]. Available: https://arxiv.org/abs/2105.05233

[21] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, D. Podell, T. Dockhorn, Z. English, K. Lacey, A. Goodwin, Y. Marek, and R. Rombach, "Scaling rectified flow transformers for high-resolution image synthesis," 2024. [Online]. Available: https://arxiv.org/abs/2403.03206

[22] T. Unterthiner, S. van Steenkiste, K. Kurach, R. Marinier, M. Michalski, and S. Gelly, "FVD: A new metric for video generation," 2019. [Online]. Available: https://openreview.net/forum?id=rylgEULtdN

[23] G. Y. Luo, G. M. Favero, Z. Luo, A. Jolicoeur-Martineau, and C. Pal, "Beyond FVD: An enhanced evaluation metrics for video generation distribution quality," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=cC3LxGZasH

[24] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," 2018. [Online]. Available: https://arxiv.org/abs/1801.03924

[25] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[26] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," 2018. [Online]. Available: https://arxiv.org/abs/1705.07750

[27] A. Bardes, Q. Garrido, J. Ponce, X. Chen, M. Rabbat, Y. LeCun, M. Assran, and N. Ballas, "Revisiting feature prediction for learning visual representations from video," 2024. [Online]. Available: https://arxiv.org/abs/2404.08471

[28] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Scholkopf, and A. Smola, "A kernel two-sample test," *J. Mach. Learn. Res.*, vol. 13, pp. 723–773, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:10742222

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[30] L. Rowe, R. Girgis, A. Gosselin, B. Carrez, F. Golemo, F. Heide, L. Paull, and C. Pal, "Ctrl-sim: Reactive and controllable driving agents with offline reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2403.19918

[31] L. Rowe, R. Girgis, A. Gosselin, L. Paull, C. Pal, and F. Heide, "Scenario dreamer: Vectorized latent diffusion for generating driving simulation environments," in *CVPR*, 2025.

[32] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov, " Large Scale Interactive Motion Forecasting for Autonomous Driving : The Waymo Open Motion Dataset ," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2021, pp. 9690–9699. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV48922.2021.00957

[33] J. Philion, X. B. Peng, and S. Fidler, "Trajeglish: Learning the language of driving scenarios," *arXiv preprint arXiv.2312.04535*, 2023.

[34] E. Vinitsky, N. Lichtlé, X. Yang, B. Amos, and J. Foerster, "Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world," 2023. [Online]. Available: https://arxiv.org/abs/2206.09889

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[36] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," 2021. [Online]. Available: https://arxiv.org/abs/2106.01345

[37] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," 2021. [Online]. Available: https://arxiv.org/abs/2106.02039

[38] N. Gontier, P. Rodriguez, I. Laradji, D. Vazquez, and C. Pal, "Language decision transformers with exponential tilt for interactive text environments," 2023. [Online]. Available: https://arxiv.org/abs/2302.05507

[39] A. Piché, R. Pardinas, D. Vazquez, and C. Pal, "A probabilistic perspective on reinforcement learning via supervised learning," in *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022.

[40] K.-H. Lee, O. Nachum, M. Yang, L. Lee, D. Freeman, W. Xu, S. Guadarrama, I. Fischer, E. Jang, H. Michalewski, and I. Mordatch, "Multi-game decision transformers," 2022. [Online]. Available: https://arxiv.org/abs/2205.15241

[41] G. Y. Luo, Z. Luo, A. Gosselin, A. Jolicoeur-Martineau, and C. Pal, "Ctrl-v: Higher fidelity autonomous vehicle video generation with bounding-box controlled object motion," *Transactions on Machine Learning Research*, 2025. [Online]. Available: https://openreview.net/forum?id=BMGikHBjlx

[42] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," 2024. [Online]. Available: https://arxiv.org/abs/2306.16927

[43] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," *CoRR*, vol. abs/2103.00020, 2021. [Online]. Available: https://arxiv.org/abs/2103.00020

[44] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *CoRR*, vol. abs/1903.11027, 2019. [Online]. Available: http://arxiv.org/abs/1903.11027

[45] S. W. Kim, J. Philion, A. Torralba, and S. Fidler, "Drivegan: Towards a controllable high-quality neural simulation," 2021. [Online]. Available: https://arxiv.org/abs/2104.15060

[46] X. Wang, Z. Zhu, G. Huang, X. Chen, J. Zhu, and J. Lu, "Drivedreamer: Towards real-world-drive world models for autonomous driving," in *European Conference on Computer Vision*. Springer, 2024, pp. 55–72.

[47] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[48] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," 2020. [Online]. Available: https://arxiv.org/abs/1805.04687

[49] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[50] A. Gosselin, G. Y. Luo, L. Lara, F. Golemo, D. Nowrouzezahrai, L. Paull, A. Jolicoeur-Martineau, and C. Pal, "Ctrl-crash: Controllable diffusion for realistic car crashes," 2025. [Online]. Available: https://arxiv.org/abs/2506.00227

[51] J. Fang, L.-l. Li, J. Zhou, J. Xiao, H. Yu, C. Lv, J. Xue, and T.-S. Chua, "Abductive ego-view accident video understanding for safe driving perception," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 22 030–22 040.

[52] Sivoha, "Car crash dataset russia 2022–2023," https://www.kaggle.com/datasets/sivoha/car-crash-dataset-russia-2022-2023, 2023, accessed: 2025-05-15.

[53] N. Liu, S. Li, Y. Du, A. Torralba, and J. B. Tenenbaum, "Compositional visual generation with composable diffusion models," 2023. [Online]. Available: https://arxiv.org/abs/2206.01714

[54] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," 2018. [Online]. Available: https://arxiv.org/abs/1611.07004

[55] P. Phunyaphibarn, P. Y. Lee, J. Kim, and M. Sung, "Unconditional priors matter! improving conditional generation of fine-tuned diffusion models," 2025. [Online]. Available: https://arxiv.org/abs/2503.20240

[56] B. Castellano, "PySceneDetect." [Online]. Available: https://github.com/Breakthrough/PySceneDetect

[57] J. Fang, D. Yan, J. Qiao, J. Xue, and H. Yu, "Dada: Driver attention prediction in driving accident scenarios," 2023. [Online]. Available: https://arxiv.org/abs/1912.12148

[58] G. Jocher, J. Qiu, and A. Chaurasia, "Ultralytics YOLO," Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[59] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer, "Sam 2: Segment anything in images and videos," 2024. [Online]. Available: https://arxiv.org/abs/2408.00714

[60] C. Li, K. Zhou, T. Liu, Y. Wang, M. Zhuang, H.-a. Gao, B. Jin, and H. Zhao, "Avd2: Accident video diffusion for accident video description," *arXiv preprint arXiv:2502.14801*, 2025.

[61] Z. Guo, Y. Zhou, and C. Gou, "Drivinggen: Efficient safety-critical driving video generation with latent diffusion models," in *2024 IEEE International Conference on Multimedia and Expo (ICME)*, 2024, pp. 1–6.

APPENDIX A LIST OF PUBLICATIONS

This thesis draws on several publications to which the author contributed during the course of the Master's program. Each chapter clearly identifies the relevant prior work on which it builds. The level of detail and amount of material presented in each case reflects both the author's involvement in the work and its importance to the overall thesis. The main contribution is the lead-author publication *Ctrl-Crash*, while the co-authored works provide important context and foundations for the thesis.

1. **Anthony Gosselin**, Ge Ya Luo, Luis Lara, Florian Golemo, Derek Nowrouzezahrai, Liam Paull, Alexia Jolicoeur-Martineau, and Christopher Pal. *Ctrl-Crash: Controllable Diffusion for Realistic Car Crashes.* arXiv preprint arXiv:2506.00227, 2025. [Currently under review].

2. Ge Ya Luo, ZhiHao Luo, **Anthony Gosselin**, Alexia Jolicoeur-Martineau, and Christopher Pal. *Ctrl-V: Higher Fidelity Autonomous Vehicle Video Generation with Bounding-Box Controlled Object Motion.* Transactions on Machine Learning Research (TMLR), 2025.

3. Luke Rowe, Roger Girgis, **Anthony Gosselin**, Bruno Carrez, Florian Golemo, Felix Heide, Liam Paull, and Christopher Pal. *CtRL-Sim: Reactive and Controllable Driving Agents with Offline Reinforcement Learning.* In Proceedings of the 8th Annual Conference on Robot Learning (CoRL), 2024

4. Luke Rowe, Roger Girgis, **Anthony Gosselin**, Liam Paull, Christopher Pal, and Felix Heide. *Scenario Dreamer: Vectorized Latent Diffusion for Generating Driving Simulation Environments.* In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2025.

The author of this thesis does not claim ownership of the primary ideas or methodological advances proposed in the co-authored works. For clarity, the author's contributions to these publications, where not listed as first author, can be summarized as follows: participation in the ideation and hypothesis development process; implementation and setup of simulation and training environments; training of model variants and development of comparative baselines; execution of evaluations and compilation of results; and contribution to writing selected sections of the manuscripts as well as the preparation of figures.

APPENDIX B ADDITIONAL RESULTS

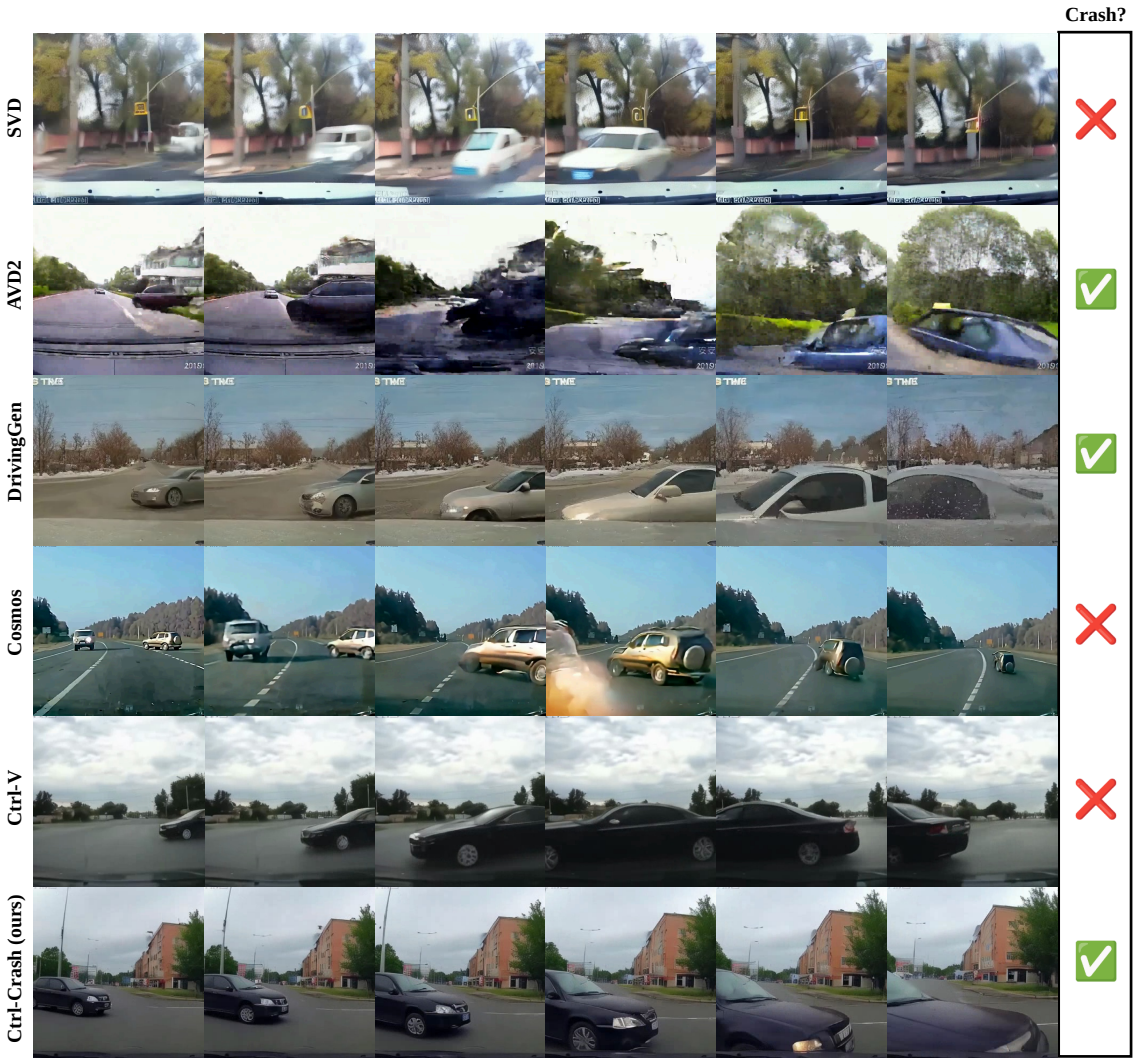


Figure B.1 **Qualitative comparison of "t-bone crashes" between different methods**. For each method, we show 6 frames from the video along with either a green check mark if there appears to have a crash in the video otherwise a red 'X'. **From top to bottom**: **SVD** (stable-video-diffusion-img2vid) [7] prompted with the initial frame from a t-bone car crash video, we see blurry vehicle a distorted motion blur as it drives in front of the ego vehicle without any collision. **AVD2** [60], we can make out what seems to be a t-bone crash with a heavily distorted black car. There are many artifacts and temporal inconsistencies which makes the sequence of events hard to follow. **DrivingGen** [61], a gray sedan drive in front of the ego vehicle progressively getting closer until it seems to collide with it. Motion is jerky and uneven between timesteps and the appearance of the gray car shapes almost every frame. **Cosmos** (Cosmos-Predict1-5B-Video2World) [8], prompted with creating a t-bone crash and 9 initial frames showing a car turn in front of the ego car, we see the leading car start to distort as the ego approaches it and then it shrivels and shrinks until it nearly disappears. **Ctrl-V** [41], prompted with a sequence of bounding-boxes suggesting a t-bone crash with a car incoming from the left, we see a car drive in from the left and then just passed the ego car without any collision. **Ctrl-Crash** (ours): prompted with the same bounding box sequence as Ctrl-V and with the discrete crash type "ego/vehicle crash", we see a physically plausible t-bone collision with the a black sedan incoming from the left.

Figure B.2 **Qualitative comparison of "rear-end crashes" between different methods.** For each method, we show 5 frames from the video along with either a green check mark if there appears to have a crash in the video otherwise a red 'X'. **From top to bottom**: **SVD** (stable-video-diffusion-img2vid) [7] prompted with the initial frame from a rear-end crash video, we see some normal driving but very inconsistent lighting and color shades with visible distorsions. **AVD2** [60], we see what appears to be a rear-end crash with a very distorted leading vehicle and background. **DrivingGen** [61], we see a rear-end crash with a leading vehicle that changes appearance every frame. Overall the video is very choppy with little temporal consistency. **Cosmos** (Cosmos-Predict1-7B-Video2World) [8], prompted with text suggesting a rear-end crash and 9 initial images where a car is rapidly approaching a truck, the predicted frames show the car unrealistically shrinking as it approaches the truck without any signs of a collision. **Ctrl-V** [41], prompted with a sequence of bounding-boxes suggesting a rear-end crash with a leading car, we see the leading car keep its distance and not crash occurs. **Ctrl-Crash** (ours): prompted with the same bounding box sequence as Ctrl-V and with the discrete crash type "ego/vehicle crash", we see a physically plausible rear-end collision with the ego vehicle visibly shaking from the impact.

APPENDIX C DATASET TYPES

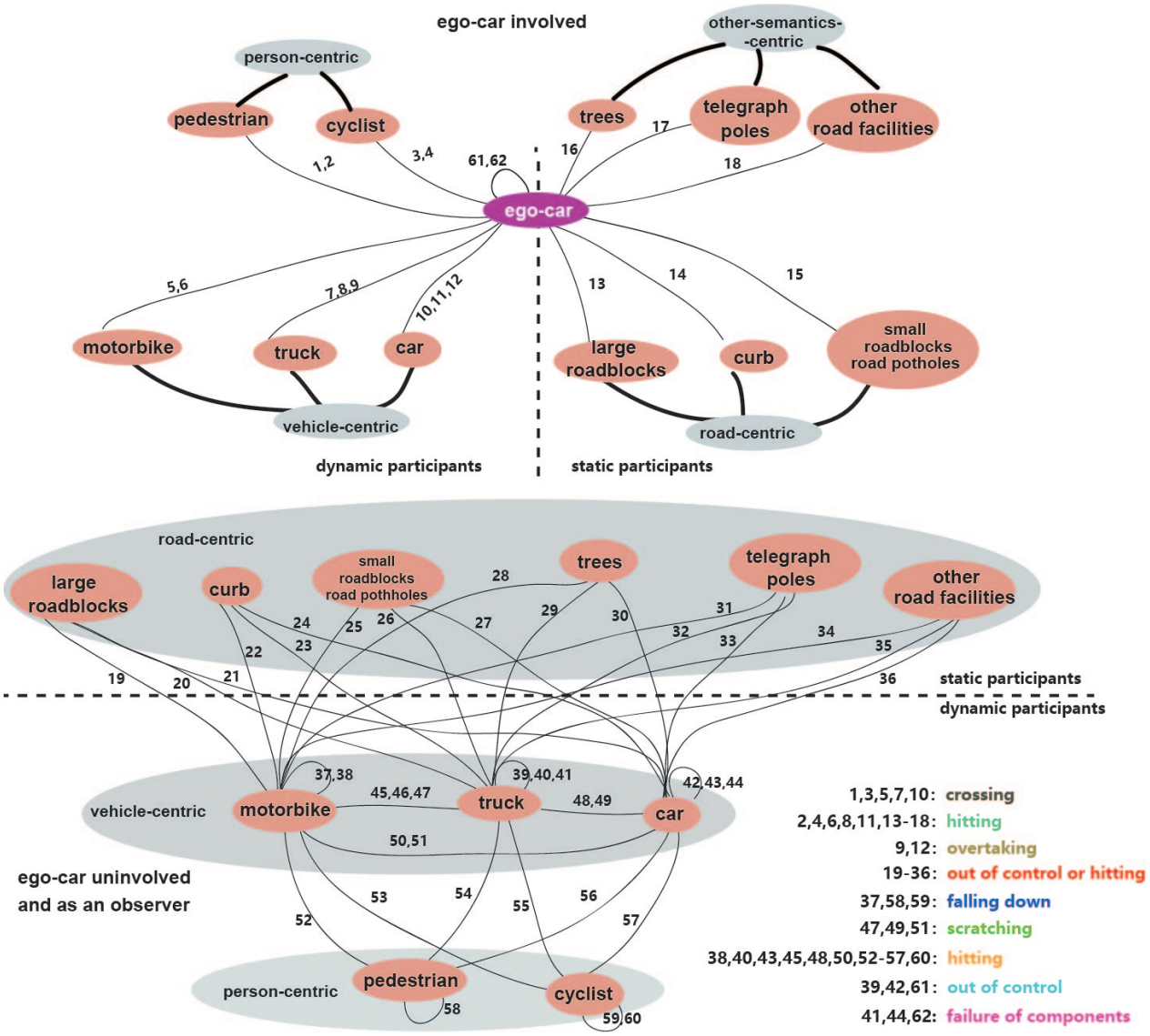


Figure C.1 **Original accident type definition used by the MM-AU dataset** [51] as defined by the DADA2000 dataset [57]

APPENDIX D DATASET ANNOTATIONS

Annotation Structure. Each annotation consists of three main fields:

- `video_source`: The filename of the source video (e.g., `"7_00951.mp4"`).
- `metadata`: High-level information about the annotated scenario, including:
  - `ego_involved` (bool): Indicates whether the ego vehicle (assumed to be the camera holder) is involved in the accident.
  - `accident_type` (int): A categorical index representing the accident type as defined by DADA2000 dataset. See Figure C.1 and Table 6.3 for more information).
- `data`: A list of per-frame annotations. Each frame entry includes:
  - `image_source`: The filename of the corresponding frame image (e.g., `"7_00951_0000.jpg"`).
  - `labels`: A list of annotated objects (bounding boxes) in the frame. Each object contains:
    - `track_id` (int): A persistent ID assigned to each object across frames.
    - `name` (str): The object class as a string (e.g., `"car"`, `"person"`, `"truck"`).
    - `class` (int): The numerical class index used internally (e.g., 0 = person, 1 = car).
    - `box` (list[float]): The bounding box coordinates, normalized to the range [0, 1], in the format `[x_min, y_min, x_max, y_max]`.