Case Study: Synthetic Driving Scenario Generation with Video Diffusion Models

Meridian Autonomy -- Generating Rare Driving Scenarios for Perception Model Training

---

Section 1: Industry Context and Business Problem

Industry: Autonomous Vehicles -- Long-Haul Freight

The autonomous trucking industry is projected to reach 87 billion USD by 2030. Unlike passenger robotaxis operating in dense urban environments, autonomous long-haul freight trucks primarily operate on interstate highways -- a more constrained driving domain, but one that introduces its own category of edge cases: dust storms across I-10 in Arizona, tire debris at highway speeds, erratic lane merges by fatigued drivers, nighttime construction zones with temporary lane markings, and animal crossings in rural stretches.

Perception systems for these trucks must reliably detect, classify, and track objects under these conditions. A single missed detection at 65 mph can be catastrophic. The safety bar is not "usually works" -- it is "works in the worst 0.01% of scenarios."

Company Profile: Meridian Autonomy

- **Founded:** 2020 in Scottsdale, Arizona
- **Employees:** ~180 (45 engineers on the perception team, 20 on simulation)
- **Funding:** Series B, USD 140M raised (lead: Founders Fund, participation from Toyota Ventures)
- **Fleet:** 45 Class 8 trucks with full sensor suites (8 cameras, 3 LiDARs, 5 radars) operating on 6 interstate corridors across Arizona, New Mexico, and Texas
- **Data pipeline:** Each truck generates approximately 1.5 TB of raw sensor data per shift. The fleet produces roughly 200 TB of driving data per month.
- **Product:** Level 4 autonomous driving for hub-to-hub freight routes. Revenue from two pilot contracts with logistics companies, generating USD 12M ARR.

Business Challenge

Meridian's perception models perform well on the 99.5% of driving that is routine -- clear weather, standard lane markings, predictable traffic. But internal safety reviews have identified 23 categories of rare scenarios where the perception stack degrades significantly:

Scenario Category	Occurrence Rate	Detection Accuracy
Clear weather, standard highway	87% of miles	99.2%
Heavy rain / spray from trucks	4% of miles	94.1%
Dust storms (desert corridors)	0.3% of miles	78.4%
Nighttime construction zones	1.2% of miles	85.7%
Tire debris / road obstacles	0.1% of miles	71.3%
Erratic merging / cut-in events	0.8% of miles	82.6%

The long tail matters. Over 500,000 miles driven per month, a 71.3% detection rate for tire debris means roughly 143 missed detections per month at highway speed. Any one of those could cause a collision.

Why It Matters

- **Safety:** Meridian needs to achieve a target detection accuracy of 95%+ across all 23 rare scenario categories to satisfy its insurance underwriter's requirements for expanded operations.
- **Regulatory:** The Arizona Department of Transportation requires demonstration of safe handling of 15 predefined edge-case categories before granting expanded corridor permits. Meridian currently passes 9 of 15.
- **Revenue:** Failing to expand corridors blocks USD 45M in contracted but deferred revenue from two additional logistics partners waiting on route availability.
- **Cost of real data:** Deliberately staging rare scenarios (e.g., driving trucks through actual dust storms with chase vehicles) costs roughly USD 18,000 per scenario-hour after vehicle, crew, safety, and insurance costs. Meridian estimates it needs 10,000 hours of rare-scenario footage. That is USD 180M at current costs -- more than the company's total funding.

Constraints

- **Compute budget:** 64 A100 GPUs available (a mix of owned and cloud reserved instances). Training must complete within 2 weeks.
- **Latency:** Generated videos are used offline for training data augmentation, so inference latency is not a hard constraint. However, generating a single 4-second clip should take under 5 minutes to allow batch generation of 10,000+ clips per week.
- **Data quality:** Generated videos must be photorealistic enough that a perception model trained on a mix of real and synthetic data does not learn artifacts. The simulation team has established a Frechet Video Distance (FVD) threshold of 250 against the real data distribution.
- **Resolution:** Dashcam footage is captured at 1280 x 720 at 10 fps. Generated videos should match this specification.

- **Privacy/compliance:** Generated videos must not reproduce identifiable faces or license plates from the training data. The company operates under SOC 2 Type II compliance.
- **Team:** The perception team has strong PyTorch experience but limited experience with diffusion models specifically. The solution must be understandable and debuggable by the team within a month.

Section 2: Technical Problem Formulation

Problem Type: Conditional Video Generation

At its core, Meridian's challenge is a conditional generation problem. Given a text description of a driving scenario (e.g., "dashcam view of a semi-truck approaching through a dense dust storm at dusk on a two-lane highway"), generate a photorealistic video clip that depicts that scenario with temporal coherence.

Why generation rather than retrieval? Because the rare scenarios by definition have insufficient real examples. You cannot retrieve what does not exist in the dataset. And why not traditional simulation (e.g., CARLA, NVIDIA Drive Sim)? Because photorealistic rendering engines require manually authored 3D assets for every scenario variant. Creating a dust storm with realistic light scattering, particle dynamics, and camera noise in a traditional simulator takes weeks of artist time per scenario. A generative model learns these visual patterns directly from data.

Why not GAN-based video generation? GANs suffer from mode collapse -- they tend to generate the same few variants of a scenario, lacking the diversity needed for robust training data augmentation. Diffusion models provide substantially better coverage of the data distribution, which is critical when the entire point is to expose the perception model to diverse edge cases.

Input Specification

The model takes two inputs:

1. **Text prompt** $c \in \mathcal{T}$: A natural language description of the desired driving scenario. The prompt follows a structured template:

```
"dashcam view of [event] during [conditions] on [road type]"
```

Examples: - "dashcam view of a tire blowout debris field during daytime on a four-lane interstate" - "dashcam view of a pickup truck making an erratic lane change during heavy rain on a two-lane highway at night"

The text is encoded into a sequence of embeddings by a pretrained text encoder (CLIP ViT-L/14): $e_{\text{text}} = \text{CLIP}(c) \in \mathbb{R}^{77\times768}$.

1. **Conditioning frame** (optional) $x_0 \in \mathbb{R}^{H\times W\times3}$: A single real dashcam image that serves as the first frame. This grounds the generation in a realistic visual context and allows control over the initial scene layout.

Output Specification

The model produces a video tensor:

$$\mathbf{v} \in \mathbb{R}^{T\times H\times W\times3}$$

with $T = 40$ frames (4 seconds at 10 fps), $H = 720$, $W = 1280$, pixel values normalized to $[0, 1]$.

Why this resolution? It matches the dashcam capture format, so synthetic and real clips can be mixed without resolution-dependent artifacts. Why 4 seconds? Internal analysis shows that 4 seconds captures the critical window for most edge-case scenarios (e.g., from first visibility of debris to the point where evasive action must begin).

Mathematical Foundation

The solution builds on three mathematical pillars. Understanding each from first principles is essential.

Pillar 1: The Diffusion Process

A diffusion model defines a forward process that gradually destroys data by adding Gaussian noise, and a reverse process that learns to reconstruct data from noise.

The forward process for a video $\mathbf{v}_0$ at diffusion timestep t is:

$$q(\mathbf{v}_t \mid \mathbf{v}_0) = \mathcal{N}(\mathbf{v}_t;\ \sqrt{\bar{\alpha}_t}\,\mathbf{v}_0,\ (1-\bar{\alpha}_t)\,\mathbf{I})$$

where $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ is the cumulative product of noise schedule coefficients.

Why does this work? The key insight is the reparameterization trick. Instead of applying noise sequentially t times, we can jump directly to any noise level:

$$\mathbf{v}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{v}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

This means the forward process is trivially parallelizable during training -- we simply sample a random t and compute $\mathbf{v}_t$ directly. No sequential computation required.

Worked example: Suppose $\bar{\alpha}_{100} = 0.25$ and a pixel in the clean video has value 0.8. Then: - Signal component: $\sqrt{0.25} \times 0.8 = 0.5 \times 0.8 = 0.4$ - Noise component: $\sqrt{1-0.25} \times \epsilon = 0.866 \times \epsilon$

The signal has been attenuated to half, and the noise dominates. By $t = T$ (the final step), $\bar{\alpha}_T \approx 0$ and the video is pure noise.

Pillar 2: Factorized Space-Time Attention

Full self-attention over a video tensor of size $T \times H \times W$ has computational complexity $O((T \cdot H \cdot W)^2 \cdot d)$, where d is the feature dimension. For our target output ($40 \times 720 \times 1280$ in pixel space, reduced to $40 \times 90 \times 160$ in latent space), full attention over 576,000 positions is computationally infeasible.

Factorized attention decomposes this into two operations:

1. **Spatial attention**: For each frame independently, attend over all spatial positions. The video features $\mathbf{h} \in \mathbb{R}^{B \times T \times (H' \times W') \times d}$ are reshaped to $\mathbb{R}^{(B \cdot T) \times (H' \times W') \times d}$. Each frame is a separate batch element. Complexity: $O(T \cdot (H' \cdot W')^2 \cdot d)$.

2. **Temporal attention**: For each spatial position independently, attend over all frames. The features are reshaped to $\mathbb{R}^{(B \cdot H' \cdot W') \times T \times d}$. Each spatial position is a separate batch element. Complexity: $O(H' \cdot W' \cdot T^2 \cdot d)$.

The total complexity is:

$$O\big(T \cdot (H'W')^2 \cdot d + H'W' \cdot T^2 \cdot d\big) \ll O\big((T \cdot H'W')^2 \cdot d\big)$$

For our latent dimensions ($T = 40$, $H' = 90$, $W' = 160$): - Full attention: $(40 \times 90 \times 160)^2 = 576{,}000^2 \approx 3.3 \times 10^{11}$ -- completely infeasible. - Spatial per frame: $(90 \times 160)^2 = 14{,}400^2 \approx 2.07 \times 10^8$, times 40 frames $\approx 8.3 \times 10^9$. - Temporal per position: $40^2 = 1{,}600$, times $14{,}400$ positions $\approx 2.3 \times 10^7$. - Total factorized: $\approx 8.3 \times 10^9$ -- roughly **40 times cheaper** than full attention.

The tradeoff is that spatial and temporal information are processed separately within each layer. An object at position (i, j) in frame t_1 that has moved to position (i', j') in frame t_2 requires multiple alternating layers of spatial and temporal attention to propagate that relationship. This is why video diffusion models stack many such blocks (typically 12-24).

Pillar 3: Latent Diffusion

Running diffusion in pixel space ($40 \times 720 \times 1280 \times 3 \approx 110$ million values) is prohibitively expensive. A Variational Autoencoder (VAE) compresses the video into a latent space:

$$\mathbf{z} = \mathcal{E}(\mathbf{v}), \quad \mathbf{v} \in \mathbb{R}^{T \times H \times W \times 3}, \quad \mathbf{z} \in \mathbb{R}^{T' \times h \times w \times c}$$

With a spatial downsampling factor of 8 and no temporal downsampling: - Input: $40 \times 720 \times 1280 \times 3 \approx 110$ M values - Latent: $40 \times 90 \times 160 \times 4 \approx 2.3$ M values - **Compression: ~48x**

The VAE is trained with a reconstruction loss and a KL divergence regularizer:

$$\mathcal{L}_{\text{VAE}} = \|\mathbf{v} - \mathcal{D}(\mathcal{E}(\mathbf{v}))\|^2 + \lambda_{\text{KL}} \cdot D_{\text{KL}}(q(\mathbf{z} \mid \mathbf{v}) \,\|\, \mathcal{N}(0, \mathbf{I}))$$

The reconstruction term ensures the latent space preserves visual information. The KL term ensures the latent distribution is smooth and well-structured, which is critical for the diffusion process -- if the latent space had irregular geometry, the Gaussian noise assumption in diffusion would break down.

Loss Function

The diffusion model is trained to predict the noise ϵ added during the forward process, conditioned on the text prompt c :

$$\mathcal{L} = \mathbb{E}_{\mathbf{z}_0, \epsilon\sim\mathcal{N}(0,\mathbf{I}), t\sim\text{Uniform}(1,T), c}\left[\left\|\epsilon - \epsilon_\theta(\mathbf{z}_t, t, c)\right\|^2\right]$$

This is a mean squared error between the true noise ϵ and the model's prediction ϵ_θ , averaged over: - Random clean latent videos $\mathbf{z}_0$ from the training set - Random noise samples ϵ - Random diffusion timesteps t - Text conditions c associated with each video

Why MSE? The noise prediction target is Gaussian, and MSE is the maximum likelihood estimator for Gaussian-distributed errors. Using L1 loss would be valid but produces slightly blurrier results because it does not penalize large errors as heavily.

Classifier-free guidance is incorporated by randomly dropping the text condition during training (replacing c with $\varnothing$ with probability $p_{\text{drop}} = 0.1$). At inference, the model produces two predictions:

$$\hat{\epsilon} = \epsilon_\theta(\mathbf{z}_t, t, \varnothing) + w \cdot \left(\epsilon_\theta(\mathbf{z}_t, t, c) - \epsilon_\theta(\mathbf{z}_t, t, \varnothing)\right)$$

where w is the guidance scale. Setting $w = 1$ recovers standard conditional generation. Setting $w > 1$ amplifies the effect of the text condition, producing videos that more closely match the prompt at the cost of reduced diversity. For synthetic training data, we use $w = 4$ -- moderate guidance that preserves diversity while ensuring scenario relevance.

Evaluation Metrics

Metric	What It Measures	Target	Business Justification
Frechet Video Distance (FVD)	Distributional similarity between real and generated videos	< 250	Below this threshold, perception models trained on synthetic data show no degradation vs. real-only training (validated internally)
Temporal Consistency Score (TCS)	Optical flow smoothness between consecutive frames	> 0.85	Videos with TCS < 0.85 cause the object tracker to produce fragmented tracks, yielding unreliable training signal
CLIP Score (text-video)	Alignment between text prompt and generated video content	> 0.28	Ensures the generated video actually depicts the requested scenario, not a generic driving clip

Metric	What It Measures	Target	Business Justification
Downstream Detection AP	Object detection mAP on a held-out real test set, after training with augmented data	> 0.95 across all 23 categories	The ultimate business metric -- does the synthetic data actually improve the perception model?

Baseline: Nearest-Neighbor Retrieval + Augmentation

The current approach retrieves the closest matching real clips from the existing dataset (using CLIP text-video similarity), then applies photometric augmentations (brightness, contrast, color jitter, synthetic rain overlay). This achieves: - CLIP Score: 0.21 (retrieved clips only approximately match the desired scenario) - Diversity: Low -- the same handful of clips get retrieved repeatedly for rare categories - Downstream AP improvement: +2.1% on rare categories (insufficient)

This baseline fails because augmentation cannot change the semantic content of a video. A color-jittered clear-weather clip does not become a dust storm.

Why Video Diffusion

1. **Temporal coherence by construction**: The diffusion model denoises the entire video jointly, producing temporally smooth outputs. This is not a post-processing step -- it is architecturally guaranteed by the factorized attention mechanism.
2. **Diversity from the generative process**: Each sample from the model is a unique video. The stochastic reverse process naturally produces diverse outputs for the same prompt.
3. **Text-conditioned generation**: The cross-attention mechanism provides fine-grained control over scenario content, enabling targeted generation of specific edge cases.
4. **Leverages pretrained knowledge**: By fine-tuning from a pretrained image diffusion model (Stable Diffusion), the model already understands photorealistic visual generation. Only the temporal layers need to be trained from scratch on driving video data.

Technical Constraints

- **Model size:** Must fit in 80 GB A100 GPU memory during inference (limits model to ~2B parameters at fp16)
- **Training compute:** 64 A100 GPUs for 14 days maximum (approximately 21,500 A100-hours)
- **Inference time:** < 5 minutes per 4-second clip at 1280 x 720
- **Training data:** 50,000 hours of real dashcam footage with structured text annotations
- **Latent space:** Spatial downsampling factor of 8 via pretrained VAE, operating on latent tensors of shape $40 \times 90 \times 160 \times 4$

Section 3: Implementation Notebook Structure

This section defines a Google Colab notebook that walks through building a simplified version of Meridian's video diffusion pipeline using real dashcam video data. The notebook uses the KITTI Raw dataset -- the industry-standard benchmark for autonomous driving perception -- with 6 driving sequences spanning city streets, residential areas, and open roads. The architectural principles -- factorized spatial-temporal attention, conditional diffusion with classifier-free guidance -- are identical to those used in production video generation systems.

Data: Real dashcam video from KITTI, resized to 64x64 RGB, extracted as 8-frame clips (0.8 seconds at 10 fps). Six driving condition labels serve as the conditioning signal, analogous to Meridian's scenario prompts (e.g., "dust storm," "nighttime construction").

Compute: All training runs on a single T4 GPU in ~75-90 minutes.

3.1 Data Acquisition: Real Dashcam Video from KITTI

The notebook downloads 6 KITTI Raw sequences directly from the KITTI S3 servers (~500 MB total). Each sequence contains PNG frames from a front-facing color camera at 10 fps:

Sequence	Driving Condition	Label
2011_09_26_drive_0005	City -- moderate traffic	0
2011_09_26_drive_0014	City -- heavy traffic, turns	1
2011_09_26_drive_0019	City -- traffic lights, intersections	2
2011_09_26_drive_0001	Residential -- parked cars, trees	3
2011_09_26_drive_0009	Residential -- narrow streets	4
2011_09_26_drive_0015	Open road -- few vehicles	5

Frames are extracted as overlapping 8-frame clips (stride 4), resized to 64x64 RGB. Students visualize real clips from each condition before building the model.

Why KITTI? It is the most widely used autonomous driving benchmark in the research community. The sequences contain real vehicles, pedestrians, cyclists, lane markings, traffic signs, and diverse road geometry -- exactly the visual elements Meridian's synthetic data must replicate.

3.2 Exploratory Data Analysis

TODO 1: Analyze Temporal Dynamics Across Driving Conditions

Students compute per-condition temporal statistics that characterize motion patterns in the real driving data. This directly informs the model: city driving has stop-and-go motion with higher frame-to-frame variance, while open road driving has steady motion with lower variance. The diffusion model must capture these differences.

```
def analyze_temporal_dynamics(videos: torch.Tensor, labels: torch.Tensor) -> dict:
    """
    Compute per-condition temporal statistics for driving video clips.

    Args:
        videos: (N, T, C, H, W) real dashcam video clips in [0, 1]
        labels: (N,) integer class labels for each clip

    Returns:
        Dictionary with keys:
            'per_condition_motion': dict mapping label (int) to mean optical flow
                magnitude (float). Proxy: mean absolute pixel difference between
                consecutive frames.
            'per_condition_brightness': dict mapping label (int) to mean pixel
                brightness (float).
            'class_counts': dict mapping label (int) to count (int).

    Steps:
        1. For each unique label:
           a. Select all clips with that label
           b. Compute frame-to-frame differences: |v[:, t+1] - v[:, t]|
              Average across all pixels, frames, and clips for that label.
           c. Compute mean brightness for that label
        2. Compute class counts
    """
```

Students then visualize the condition distribution and per-condition motion -- this is the long-tail problem. Some conditions have far more clips than others, directly mirroring Meridian's challenge where rare scenarios are underrepresented.

3.3 Baseline: Frame-Independent Generation

Before building the full video model, students demonstrate why frame-independent generation fails. A simple 2D U-Net image diffusion model is trained on individual KITTI frames, then used to generate 8-frame clips by sampling each frame independently.

Provided: A `SimpleImageUNet` (2D U-Net) and `cosine_noise_schedule` function. The model trains for 5 epochs on all individual dashcam frames.

TODO 2: Generate Videos Frame-by-Frame and Measure Coherence

Students implement the DDPM reverse loop for independent frame generation, then measure temporal coherence (cosine similarity between consecutive frames) on both real and independently generated clips.

```
def generate_independent_video(model, noise_schedule, n_frames=8, size=64):
    """
    Generate a video by sampling each frame independently from an image model.
    Returns: (n_frames, 3, size, size) in [0, 1]

    Steps:
        1. For each frame: sample x_T ~ N(0,I), run DDPM reverse loop
        2. Stack and clamp to [0, 1]
    """

def temporal_coherence_score(video: torch.Tensor) -> float:
    """
    Mean cosine similarity between consecutive flattened frames.
    Returns: float in [-1, 1], higher = more temporally coherent.
    """
```

The verification cell compares 5 real vs. 5 independently generated clips. The gap quantifies why Meridian cannot use frame-by-frame generation: a perception model trained on temporally incoherent video would learn fragmented object tracks.

3.4 Model Design: Factorized Video Diffusion

The core architecture has three components: (1) a multi-head self-attention module reusable for both spatial and temporal attention, (2) factorized attention wrappers that handle the axis reshaping, and (3) a conditional video U-Net that combines 2D convolutions (per-frame) with factorized attention for temporal coherence.

Provided: `MultiHeadSelfAttention` module, `FactorizedSpaceTimeBlock` (spatial -> temporal -> FFN), and `ConditionalVideoUNet` with 2D conv encoder/decoder, condition embedding for 6 driving conditions, and factorized attention at the bottleneck.

TODO 3: Spatial Attention Block

```
class SpatialAttentionBlock(nn.Module):
    """Self-attention within each frame independently.
    Key reshape: (B, T, H, W, C) -> (B*T, H*W, C)
    Each frame is independent -- frame 1 cannot see frame 2."""

    def forward(self, x):
        """
        Steps:
            1. Save residual, apply layer norm
            2. Reshape to (B*T, H*W, C)
            3. Apply attention, reshape back
            4. Add residual
        """
```

TODO 4: Temporal Attention Block

```
class TemporalAttentionBlock(nn.Module):
    """Self-attention across frames at each spatial position.
    Key reshape: (B, T, H, W, C) -> (B*H*W, T, C)
    Each spatial position attends across all frames."""

    def forward(self, x):
        """
        Steps:
            1. Save residual, apply layer norm
            2. Permute to (B, H, W, T, C), reshape to (B*H*W, T, C)
            3. Apply attention
            4. Reshape back to (B, T, H, W, C), add residual
        """
```

Verification cells check output shapes and confirm spatial independence (modifying frame 2 must not affect the output for frame 0).

Thought questions: - Why process spatial attention first, then temporal? Would reversing the order change behavior? - What information cannot be captured by factorized attention that full 3D attention could?

3.5 Training Strategy

Optimizer and schedule: - AdamW with weight decay 0.01 - Learning rate: 3e-4 with cosine decay to 1e-5 over 40 epochs - Gradient clipping at norm 1.0

Why AdamW over SGD? Video diffusion models have highly non-stationary gradients due to random timestep sampling -- different timesteps produce gradients of vastly different magnitudes. Adam's per-parameter adaptive learning rates handle this naturally. Weight decay prevents memorizing specific video clips.

TODO 5: Training Step with Classifier-Free Guidance

```
def training_step(model, videos, labels, noise_schedule, p_uncond=0.1):
    """
    One training step of conditional video diffusion.

    Args:
        model: ConditionalVideoUNet
        videos: (B, C, T, H, W) clean dashcam clips
        labels: (B,) driving condition labels (0 to N_CLASSES-1)
        noise_schedule: Dict with 'alphas_cumprod' on correct device
        p_uncond: Probability of dropping condition (CFG)

    Returns:
        Scalar loss tensor

    Steps:
        1. Sample random timesteps t ~ Uniform(0, num_timesteps-1)
        2. Sample noise eps ~ N(0, I)
        3. Compute noisy videos: v_t = sqrt(alpha_bar_t) * v_0 + sqrt(1 - alpha_bar_t) * eps
           Reshape alpha_bar_t to (B, 1, 1, 1, 1) for broadcasting.
        4. CFG dropout: clone labels, set labels[mask] = -1 where mask = (torch.rand(B) < p_uncond)
        5. Predict noise: eps_pred = model(v_t, t, dropped_labels)
        6. Return F.mse_loss(eps_pred, eps)
    """
```

The full training loop runs 40 epochs on real KITTI data with batch size 8. A training loss curve is plotted afterward.

Thought question: What happens if you set `p_uncond = 0` (never drop the condition)? The model still generates conditioned videos, but classifier-free guidance at inference will not work. Why?

3.6 Generation and Evaluation

Provided: Conditional sampling function with classifier-free guidance (`sample_videos`), which generates clips by running the DDPM reverse loop with guided noise prediction: `eps = eps_uncond + w * (eps_cond - eps_uncond)`.

Students generate 3 clips per driving condition and compare them side-by-side with real KITTI clips.

TODO 6: Frechet Video Distance

FVD is Meridian's primary quality metric. Students compute it using a provided 3D CNN feature extractor (fixed random weights for reproducibility).

```
def compute_fvd(real_videos, gen_videos, feature_extractor):
    """
    Compute Frechet Video Distance.

    Args:
        real_videos: (N, C, T, H, W) real clips
        gen_videos: (N, C, T, H, W) generated clips
        feature_extractor: Maps (B, C, T, H, W) -> (B, feat_dim)

    Returns:
        FVD score (float, lower = better). Target: < 250.

    Steps:
        1. Extract features for real and generated videos in batches
        2. Compute mu and covariance for each set
        3. FVD = ||mu_r - mu_g||^2 + Tr(S_r + S_g - 2 * sqrtm(S_r @ S_g))
    """
```

3.7 Error Analysis: Driving-Specific Failure Modes

TODO 7: Categorize Failures in Generated Driving Clips

At Meridian, every generated clip passes through an automated quality gate. Students implement driving-specific failure detection:

```
def driving_error_analysis(generated_clips, condition_labels, n_inspect=18):
    """
    Categorize driving-specific failure modes.

    Returns:
        Dict mapping failure_type to count:
            'temporal_flicker': max frame-to-frame pixel diff > 0.4
            'static_video': mean frame-to-frame diff < 0.01
            'color_drift': mean color of first vs last frame differs by > 0.15
            'clean': no failures detected

    Steps:
        1. Collect clips, randomly select n_inspect
        2. For each: compute frame diffs and color drift
        3. Categorize based on thresholds
    """
```

Thought questions: Each failure mode has different downstream impact: - *Temporal flicker* causes the object tracker to fragment, losing track continuity - *Static video* provides no motion learning signal (the perception model sees a photograph) - *Color drift* teaches the model to expect impossible lighting changes

Which would you prioritize fixing first? What architectural change would address it?

3.8 Inference Benchmarking

TODO 8: Profile Generation Latency

Meridian needs to generate 10,000 clips per week. Students profile the generation pipeline.

```
def benchmark_generation(model, noise_schedule, n_runs=3):
    """
    Returns: dict with 'mean_seconds', 'std_seconds', 'per_step_ms', 'peak_memory_mb'

    Steps:
        1. Warm up with 1 dummy generation
        2. Time n_runs with torch.cuda.synchronize()
        3. Track peak GPU memory
    """
```

The verification cell computes how many GPUs Meridian would need running 24/7 and lists optimization paths: DDIM (50 steps instead of 200), half-precision (fp16), model distillation, batch generation.

3.9 Ethical and Bias Analysis

TODO 9: Per-Condition Quality Audit

If the model generates "open road" scenarios with lower quality than "city" scenarios, the perception model trained on that data will have a systematic blind spot on highway routes -- exactly where Meridian's trucks operate most.

```
def per_condition_quality_audit(model, noise_schedule, n_per_class=5):
    """
    Audit generation quality bias across driving conditions.

    Returns:
        Dict with:
            'per_condition_coherence': {cls: float}
            'per_condition_sharpness': {cls: float}
            'worst_condition': int
            'best_condition': int
            'quality_gap': float

    Steps:
        1. For each condition, generate n_per_class clips
        2. Compute temporal coherence and sharpness (Sobel gradient magnitude)
        3. Identify best/worst conditions and gap
    """
```

Critical questions for Meridian's ML team: - If the worst-quality condition corresponds to the rarest driving scenario, how does this amplify the long-tail safety problem? - How would you enforce that ALL conditions meet a minimum quality threshold before synthetic data enters the training pipeline? - What SOC 2 controls would you place around the synthetic data pipeline? Consider: data provenance tracking, model versioning, and audit logs.

Scaling to Production

The notebook concludes with guidance for bridging the gap from 64x64/8-frame prototypes to Meridian's production system (1280x720/40 frames): - **Latent diffusion:** Compress frames to 16x smaller latent space via pretrained VAE - **Cascaded upsampling:** Low-res generation + super-resolution (Imagen Video approach) - **Pretrained initialization:** Start from Stable Diffusion's spatial layers, train only temporal layers - **Larger datasets:** nuScenes (1000 scenes, rain/night), BDD100K (100K clips with weather labels), Waymo Open

Section 4: Production and System Design Extension

Architecture Overview

The production system at Meridian consists of six major components:

```
[Prompt Service] --> [Diffusion Inference Service] --> [Quality Gate]
       |                       |                           |
  [Prompt DB]         [Model Registry]          [Data Lake]
                                                      |
                                              [Perception Training Pipeline]
```

1. **Prompt Service:** Accepts scenario generation requests via API, validates prompts against a schema, and queues them for batch processing.
2. **Diffusion Inference Service:** Runs the video diffusion model on a GPU cluster. Handles batching, DDIM acceleration, and classifier-free guidance.
3. **Quality Gate:** Runs automated quality checks (FVD, temporal coherence, CLIP score) on generated videos. Rejects videos that fall below thresholds.
4. **Prompt DB:** Stores the full history of prompts, generation parameters, and quality scores for reproducibility.
5. **Model Registry:** Stores versioned model checkpoints, training configs, and evaluation reports.
6. **Data Lake:** Stores accepted generated videos alongside real data, with metadata tags distinguishing synthetic from real.

API Design

Generate Video Endpoint

```
POST /api/v1/generate
```

Request:

```
{
  "prompt": "dashcam view of tire debris on interstate during daytime",
  "conditioning_image_url": "s3://meridian-data/frames/000123.png",
  "num_frames": 40,
  "guidance_scale": 4.0,
  "seed": 42,
  "priority": "normal"
}
```

Response:

```
{
  "job_id": "gen-2024-0817-001234",
  "status": "queued",
  "estimated_completion": "2024-08-17T14:35:00Z",
  "callback_url": "https://api.meridian.ai/v1/jobs/gen-2024-0817-001234"
}
```

Retrieve Result Endpoint

```
GET /api/v1/jobs/{job_id}
```

Response:

```
{
  "job_id": "gen-2024-0817-001234",
  "status": "completed",
  "video_url": "s3://meridian-synth/videos/gen-2024-0817-001234.mp4",
  "quality_scores": {
    "fvd": 198.3,
    "temporal_coherence": 0.91,
    "clip_score": 0.31
  },
  "metadata": {
    "model_version": "v2.3.1",
    "inference_time_seconds": 187.4,
    "gpu_type": "A100-80GB"
  }
}
```

Serving Infrastructure

- **Compute:** 8x A100-80GB nodes behind a Kubernetes autoscaler. Each node generates one video at a time (the model occupies the full GPU).
- **Queue:** Redis-backed priority queue. Safety-critical scenario requests (from the perception team's active investigation list) get elevated priority.
- **Batch processing:** During off-peak hours (nights, weekends), the cluster runs batch generation jobs from a weekly scenario plan.
- **Storage:** Generated videos are written to S3 with lifecycle policies -- raw outputs retained for 90 days, quality-passed videos retained indefinitely.

Latency Budget

Component	Target	Notes
Prompt encoding (CLIP)	50 ms	Single GPU forward pass
VAE encoding (if conditioning image)	200 ms	Per-frame encoding
Diffusion reverse process (50 DDIM steps)	180 s	Bulk of computation
VAE decoding	5 s	Decode all 40 latent frames
Quality gate checks	10 s	FVD approximation + coherence
Video encoding (H.264)	3 s	ffmpeg on CPU
Total	**~200 s**	Well within 5-minute budget

Monitoring

Metrics to track: - **Generation throughput:** Videos generated per hour, per node - **Quality score distributions:** Rolling 24-hour histograms of FVD, TCS, CLIP score - **Rejection rate:** Percentage of generated videos failing the quality gate (alert if > 15%) - **GPU utilization:** Target > 85% during batch jobs - **Prompt coverage:** Number of unique scenario categories generated per week (target: all 23) - **Downstream impact:** Weekly perception model mAP on the rare-scenario test set

Alerting thresholds: - FVD 24-hour mean > 300: Investigate potential model degradation - Rejection rate > 20%: Halt generation, investigate - GPU utilization < 50% for > 2 hours: Scheduling issue - Any scenario category with 0 generated videos in 7 days: Coverage gap

Model Drift Detection

The primary drift concern is between the real dashcam data distribution and the model's learned distribution. As Meridian's fleet expands to new corridors (e.g., Pacific Northwest with very different weather patterns), the model may generate scenarios that do not match the new operational domain.

Detection approach: 1. Monthly FVD evaluation against a rolling 30-day sample of real fleet data 2. CLIP embedding clustering -- compare cluster centroids between real and generated distributions 3. Downstream perception model performance on real data, segmented by whether training included synthetic data for that scenario category

Mitigation: When drift is detected, trigger a fine-tuning run on the latest 30 days of real data. The temporal layers are most sensitive to distributional shift and should be fine-tuned first.

Model Versioning

- **Registry:** MLflow Model Registry with promotion stages: Development -> Staging -> Production
- **Naming convention:** `video-diffusion-v{major}.{minor}.{patch}` (e.g., `v2.3.1`)
- Major: Architecture change
- Minor: Retraining with new data or hyperparameters
- Patch: Inference optimization (quantization, distillation) without retraining
- **Rollback:** Previous production model is retained for 30 days. Rollback is a single API call that updates the load balancer's model endpoint.
- **Artifact storage:** Each version stores: checkpoint, training config, evaluation report, a fixed set of 100 generated samples for visual comparison.

A/B Testing

When deploying a new model version, Meridian uses an A/B test against the production model:

1. **Traffic split:** 10% of generation requests routed to the candidate model, 90% to production
2. **Duration:** 7 days minimum (to accumulate sufficient samples)
3. **Primary metric:** Downstream perception model mAP improvement when trained with candidate-generated data vs. production-generated data
4. **Guardrail metrics:** FVD must not regress by more than 10%. Rejection rate must not increase by more than 5 percentage points.

5. **Statistical significance:** Two-sided t-test with $p < 0.05$ and minimum effect size of 1% mAP improvement
6. **Decision:** Promote if primary metric improves significantly and no guardrail is breached. Otherwise, iterate on the candidate.

CI/CD for ML

```
[Code Change / New Data] --> [Unit Tests] --> [Small-Scale Training (1 GPU, 1 epoch)]
        --> [Quality Gate on Validation Set] --> [Full Training (64 GPUs)]
        --> [Evaluation Suite] --> [A/B Test Deployment] --> [Production Promotion]
```

- **Triggers:** Code changes trigger the full pipeline. New data (monthly) triggers from the full training stage onward.
- **Unit tests:** Model forward pass, loss computation, sampling loop correctness, data loading
- **Small-scale gate:** If validation FVD after 1 epoch on 1 GPU exceeds 500, abort (likely a bug, not a scaling issue)
- **Evaluation suite:** FVD, TCS, CLIP score, downstream perception mAP on 5 scenario categories, inference latency
- **Artifacts:** Every pipeline run produces a versioned evaluation report and sample videos

Cost Analysis

Item	Specification	Monthly Cost
Training (initial)	64x A100 for 14 days	~USD 150,000 (one-time)
Training (monthly fine-tune)	16x A100 for 3 days	~USD 14,000
Inference cluster	8x A100, ~50% utilization	~USD 48,000
Storage (S3)	~50 TB generated video + metadata	~USD 1,200
Quality gate compute	CPU instances for FVD/TCS	~USD 500
MLflow / monitoring	Managed services	~USD 300
Monthly operational total		**~USD 64,000**

Compare this to the USD 180M cost of collecting real rare-scenario data. At USD 64,000/month, the synthetic data pipeline pays for itself if it replaces even 0.5% of real data collection needs -- which it does many times over.

References

- Ho et al., "Video Diffusion Models" (2022)
- Ho et al., "Imagen Video: High Definition Video Generation with Diffusion Models" (2022)
- Blattmann et al., "Stable Video Diffusion: Scaling Latent Video Diffusion Models to Large Datasets" (2023)

- Peebles and Xie, "Scalable Diffusion Models with Transformers" (2023)
- Hu et al., "GAIA-1: A Generative World Model for Autonomous Driving" (2023)
- Kim et al., "DriveGAN: Towards a Controllable High-Quality Neural Simulation" (2021)
- Rombach et al., "High-Resolution Image Synthesis with Latent Diffusion Models" (2022)
- Unterthiner et al., "Towards Accurate Generative Models of Video: A New Metric and Challenges" (2019) -- FVD metric