Data Engines for Autonomous Vehicles: A Reference Architecture

Anjan Goswami

December 2025

Abstract. This paper presents a reference architecture for production data engines in autonomous vehicle systems—the closed-loop infrastructure that transforms raw sensor data into training signal for ML models. Drawing on experience building data engines for healthcare AI, document intelligence, and search systems, I describe how these patterns transfer to embodied AI. The architecture has evolved through three distinct eras: human-labeled real data (pre-2017), simulation-augmented pipelines (2017–2021), and foundation model integration (2022–present). Modern data engines operate dual loops of real and synthetic data, employ model-aware mining strategies, leverage VLMs for scalable labeling, and use neural world models for closed-loop evaluation. I provide concrete interface specifications and discuss failure modes, cross-domain patterns, and the role of open datasets in accelerating research.

1. Introduction

I have spent the past several years architecting data engines—the closed-loop systems that turn raw operational data into training signal for ML models. I have built these for search ranking, virtual assistants, recommender systems, and healthcare AI. The pattern is consistent: ingest, triage, mine, label, train, evaluate, deploy, repeat.

What strikes me now is how dramatically this pattern has evolved. When I first encountered data engines around 2018, they were fundamentally about *human labeling at scale*—the bottleneck was getting enough annotations fast enough. Today's data engines look radically different. They are as much about *generating* data as collecting it, about *model-aware curation* rather than uniform coverage, and about *foundation models* that change the entire economics of what needs labeling.

I think of this evolution in three eras:

Era 1 (Pre-2017): Real Data + Human Labels. The original data engine was simple: collect real-world data, have humans label it, train models. Tesla's early autopilot, Waymo's original labeling pipelines, academic benchmarks like KITTI (2012). The constraint was annotation throughput—every additional training example required human hours. The long tail was effectively unreachable.

Era 2 (2017–2021): Simulation Rises. CARLA (November 2017) and domain randomization techniques (Tobin et al., March 2017) introduced synthetic data as a serious complement to real data. Open datasets (nuScenes March 2019, Waymo Open Dataset August 2019) accelerated research. Tesla's AI Day 2021 revealed a mature data engine with 1000+ in-house labelers, auto-labeling infrastructure, and simulation.

Era 3 (2022–Present): Foundation Models Change Everything. CLIP (January 2021) enabled vision-language supervision. ChatGPT (November 2022) made LLM capabilities mainstream. World models like GAIA-1 (June 2023) began generating photorealistic driving video. Neural sensor simulation (UniSim, CVPR 2023) enabled closed-loop testing from real logs. The labeling economics inverted: foundation models can now produce labels at pennies per example that previously cost dollars.

This paper reflects the 2024–2025 state of the art.

2. The Dual Loop

Modern data engines operate two parallel loops that feed each other:

Real Data Loop: Deploy → Collect → Mine → Label → Train → Evaluate → Deploy

Synthetic Data Loop: Scenario Specification → Generation → Validation → Training → Evaluation → Scenario Refinement

These are tightly coupled. Real data reveals coverage gaps that synthetic generation fills. Synthetic scenarios get validated against real-world distribution. Models trained on synthetic data are evaluated on real benchmarks.

The balance has shifted dramatically. In 2018, synthetic data might constitute 5–10% of training examples. By 2024, it is plausibly 30–50% for some model types—especially for rare events, safety-critical scenarios, and sensor configurations that do not exist in your fleet yet.

Principle: Real data grounds you in reality; synthetic data lets you reach the long tail. You need both.

3. Vehicle Layer

The on-vehicle system follows: Sensors → Time Sync → Inference + Triggers → Buffer → Upload Queue.

3.1 Sensors

The vehicle carries a sensor suite providing 360-degree coverage:

- **Cameras** (typically 8): RGB at 30fps, H.265 encoded
- **LiDAR** (1–5 units): 3D point clouds at 10–20Hz
- **Radar** (4–6 units at 77GHz): Doppler velocity
- **IMU**: 200Hz sampling for motion compensation
- **GNSS**: RTK corrections for cm-level positioning

3.2 Inference and Triggers

The on-vehicle compute runs production models in real-time. A trigger detector monitors for scenarios worth uploading: model uncertainty exceeding threshold, time-to-collision under 3 seconds, novel objects (OOD detection), driver disengagement, safety-critical maneuvers, and random sampling at 1% baseline.

3.3 Buffer and Upload

The vehicle cannot upload everything—at 2GB/s raw sensor rate, that would be 7TB per hour. Instead, it maintains a rolling buffer (5 min, ∼600GB) and extracts triggered snapshots (-30s to $+10$s around each trigger).

Principle: The vehicle decides what is interesting. Collection strategy determines everything downstream.

4. Cloud Ingestion

Upload → Integrity Check → Calibration → Segment Builder → Message Queue

4.1 Upload and Validation

The upload service handles resumable transfers (vehicles lose connectivity), deduplication by content hash, and bandwidth management across the fleet. Integrity checking validates completeness via SHA256 checksums, counts frames to detect gaps, and flags corrupted data for exclusion.

4.2 Calibration

Raw sensor data requires calibration: camera intrinsics (focal length, distortion), sensor extrinsics (position/orientation relative to vehicle frame), LiDAR-camera alignment, and temporal offset correction. Calibration parameters come from periodic depot routines and drift over time. They must be versioned and tracked.

4.3 Segment Builder

The segment builder assembles raw streams into canonical **Segments**—the atomic unit of data in the system. A Segment contains:

- Segment ID (UUID) and vehicle ID
- Timestamp range (start/end in nanoseconds)
- Camera frames (synchronized, rectified)
- LiDAR sweeps (motion-compensated)
- Radar returns (ego-velocity corrected)
- Vehicle state (pose, velocity, acceleration)
- Map context (HD map crop around trajectory)
- Triggers (why this segment was uploaded)
- Calibration bundle (all parameters at collection time)

Segments are typically 20–40 seconds—enough context for a scenario, small enough to process efficiently.

4.4 Message Queue

Completed segments publish to Kafka (topic: `segments.raw`, partitioned by `vehicle_id`, 7-day retention) for downstream consumers.

5. Storage and Indexing

Object Store + Metadata DB + Catalog + Multimodal Embeddings + Vector Index

5.1 Object Store

Raw and processed data lives in cloud storage (S3/GCS). Directory structure separates raw (immutable) from processed (append-only):

- `/raw/{date}/{vehicle}/{segment}/` — original sensor data, never modified
- `/processed/{date}/{segment}/` — labels, embeddings, derived artifacts
- `/synthetic/{experiment}/{scenario}/` — generated data with provenance

Storage costs dominate the data engine budget. A 1,000-vehicle fleet uploading aggressively generates petabytes per month. Lifecycle policies move old data to cold storage after 90 days.

5.2 Metadata Database

PostgreSQL stores queryable metadata: segment records (ID, vehicle, timestamp, triggers, status, storage path), label records (segment, frame, label type, source, confidence, creator), synthetic data records (generator version, scenario parameters, validation status), and dataset records (version, segment list, split assignments).

5.3 Analytical Catalog

Iceberg or Delta Lake provides versioned, transactional access to analytical tables (segments, frames, objects, trajectories, labels, training runs, model versions). The catalog enables time travel, schema evolution, and full lineage tracking.

5.4 Multimodal Embeddings and Tagging

As segments arrive, a feature extraction pipeline computes derived representations:

- **Visual embeddings:** Scene-level representations from frozen encoders (CLIP, SigLIP, DINOv2)
- **BEV embeddings:** Bird's-eye view representations from BEVFormer-style encoders
- **Language descriptions:** VLM-generated natural language scene descriptions (used for text-based retrieval)
- **Scenario tags:** Rule-based and learned classifiers for weather, lighting, road type, traffic density, agent types, events, anomalies

The language descriptions are a 2023+ addition. Given a driving clip, a VLM generates descriptions like "Night urban intersection, light rain, pedestrian crossing against signal, delivery truck blocking right lane." These enable natural-language search over the entire database.

5.5 Vector Index

Embeddings are indexed (Milvus, Pinecone, or FAISS) for similarity search. Given a seed scenario, you can retrieve the k most similar segments across the entire database in seconds. Language descriptions enable hybrid search: "Find scenes similar to this embedding AND matching 'construction zone at night'."

5.6 Structured Query Layer

Vector search is powerful but insufficient alone. Production data engines require traditional structured queries:

SQL-based retrieval: Most data access patterns are structured queries over metadata—"all segments from vehicle X in the last 7 days," "segments with pedestrian labels and rain tags," "segments that triggered on model uncertainty above 0.8." PostgreSQL or a data warehouse (Snowflake, BigQuery, Databricks) handles these efficiently.

Metadata filtering: Before expensive embedding search, filter by structured attributes: date range, vehicle ID, geographic region, trigger type, label status. This reduces the search space by orders of magnitude.

Full-text search: Elasticsearch or similar indexes the VLM-generated language descriptions for keyword search. Useful for quick lookups: "find segments mentioning 'construction' or 'debris'."

Composite queries: Real mining queries combine all three: structured filters (date, region) $\rightarrow$ full-text match (scenario keywords) $\rightarrow$ embedding similarity (visual similarity to seed). The query planner optimizes execution order.

Materialized views: Pre-computed aggregations for common queries—segments per scenario type, label coverage by geography, trigger distribution over time. Updated incrementally as new data arrives.

Principle: Everything becomes a queryable scenario. If you cannot find data by scenario type, you cannot systematically improve on that scenario type.

6. Model-Aware Mining

Model Architecture $\rightarrow$ Data Requirements $\rightarrow$ Gradient-Based Importance $\rightarrow$ Per-Task Mining $\rightarrow$ Priority Ranker $\rightarrow$ Labeling Queue

Mining is the heart of the data engine. The 2024 insight is that *different model architectures have fundamentally different data needs*. A camera-only BEV transformer needs different data than a LiDAR-primary detector. An end-to-end model needs different data than a modular perception stack. Mining must be model-aware.

6.1 Model Types and Data Requirements

Table 1 summarizes how different model architectures require different mining strategies.

Table 1: Model Types and Data Requirements

Model Type	Architecture	Data Needs	Mining Strategy
Detection (3D boxes)	PointPillars, CenterPoint	Dense object diversity, occlusion patterns	Class-balanced sampling, hard negative mining
Segmentation	U-Net variants, BEV grids	Pixel-level boundary precision	Edge case scenes, rare surface types
Occupancy	3D voxel networks	Volumetric completeness	Complex geometry, multi-object occlusion
BEV Perception	BEVFormer, BEVDet	Multi-camera consistency, temporal coherence	View synthesis failures, tracking edge cases
End-to-End	UniAD, VAD	Full scenario diversity, planning failures	Intervention mining, trajectory divergence
World Models	GAIA, DriveDreamer	Scene dynamics, multi-agent interaction	Rare interactions, physics edge cases
VLM-based	LLM + visual encoder	Language-groundable scenarios	Scenes with complex semantics, unusual objects

6.2 Gradient-Based Importance

For a specific model checkpoint, you can compute which examples most affect its parameters:

- **Influence functions:** Approximate the effect of removing a training example on model predictions
- **Gradient magnitude:** Examples with high gradient norm are "actively being learned"
- **Loss curvature:** Examples where the loss surface is steep indicate learning opportunities

These techniques help identify what the *current model* needs to improve, rather than sampling uniformly.

6.3 Per-Task Failure Analysis

Modern AV stacks have multiple heads sharing a backbone. Mining should analyze failures per head:

- Detection head failing on small objects $\rightarrow$ mine segments with distant/small objects
- Lane prediction head failing on intersections $\rightarrow$ mine complex intersection geometry
- Trajectory prediction head failing on cyclists $\rightarrow$ mine cyclist interaction scenarios

6.4 Foundation Model Fine-Tuning vs. Training from Scratch

The mining strategy depends on whether you are fine-tuning a pretrained model or training from scratch:

- **Fine-tuning:** Focus on domain-specific edge cases; the model already has strong priors
- **From scratch:** Need broader coverage of basic scenarios first, then edge cases
- **Continual learning:** Mine for distribution shift, catastrophic forgetting prevention

Principle: Mine for what the model needs, not what you have. Model-aware mining is the key differentiator in 2024+.

7. Active Learning and Data-Centric AI

Model Uncertainty $\rightarrow$ Sample Selection $\rightarrow$ Human Labeling $\rightarrow$ Model Update $\rightarrow$ Iterate

Active learning closes the loop between model performance and data selection. Instead of randomly sampling data for labeling, the model itself identifies which examples would be most valuable to learn from. This is the core of data-centric AI: improving models by systematically improving data rather than just scaling model size.

7.1 Uncertainty Sampling

The simplest active learning strategy: label examples where the model is least confident.

Entropy-based: Select examples with highest prediction entropy $H = -\sum p_i \log p_i$. High entropy means the model is uncertain across multiple classes.

Margin sampling: Select examples where the difference between the top two predictions is smallest. The model is torn between alternatives.

Least confidence: Select examples where $\max(p_i)$ is lowest. The model has no strong prediction.

For AV perception, uncertainty manifests as: detection scores near threshold, inconsistent predictions across frames, disagreement between camera and LiDAR detections.

7.2 Query-by-Committee

Train multiple models (or use dropout at inference to simulate an ensemble). Select examples where the committee disagrees most.

Implementation: Maintain 3–5 model checkpoints from different training runs or epochs. For each unlabeled segment, measure prediction variance across the committee. High variance indicates the models learned different decision boundaries—this is where more data helps.

AV application: Run the production model and a challenger model on the same segment. Disagreement on object existence, classification, or trajectory prediction flags high-value labeling candidates.

7.3 Expected Model Change

Select examples that would cause the largest gradient update if labeled. This directly targets data that changes the model.

Gradient magnitude: Examples with high $\|\nabla_\theta L(x, y)\|$ are "surprising" to the model—far from what it expects.

Expected gradient length (EGL): For unlabeled data, estimate the expected gradient over possible labels. Select examples with highest expected impact.

Influence functions: Approximate how adding an example to training would affect predictions on a validation set. Computationally expensive but principled.

7.4 Diversity-Aware Selection

Pure uncertainty sampling can oversample similar edge cases. Combine uncertainty with diversity:

Coreset selection: Choose examples that best represent the unlabeled pool. Maximize coverage of the feature space.

Clustering + uncertainty: Cluster unlabeled data by embedding, then select the most uncertain example from each cluster.

Batch-aware selection: When selecting a batch for labeling, penalize redundancy. Use determinantal point processes (DPPs) or facility location objectives to encourage diversity.

7.5 Cold Start and Warm Start

Cold start: When beginning a new task or domain with no labeled data, active learning cannot leverage model uncertainty. Use diversity sampling (cover the feature space) or leverage pretrained embeddings to identify representative examples.

Warm start: With an existing model, active learning accelerates improvement. Each labeling batch is targeted at current weaknesses.

Curriculum integration: Start with high-confidence examples (easy to label correctly, establishes baseline), then shift to uncertain examples (harder, higher value).

7.6 Human-in-the-Loop Integration

Active learning requires tight integration with the labeling pipeline:

Prioritized queues: Labeling interface pulls from uncertainty-ranked queue, not random sample. Human effort goes to highest-value examples.

Rapid iteration: Model retrains frequently (daily or after each labeling batch) so uncertainty estimates stay current. Stale uncertainty scores waste labeling budget.

Labeler feedback: When labelers flag ambiguous or hard examples, feed this back to boost priority for similar cases.

Cost-aware selection: Some labels are expensive (full 3D annotation) vs. cheap (binary verification). Active learning should account for labeling cost, not just information gain.

7.7 Data-Centric AI Principles

Active learning is one component of the broader data-centric AI paradigm:

Data quality over quantity: A smaller, well-curated dataset often outperforms a larger noisy one. Invest in label quality, consistency, and coverage.

Systematic error analysis: Slice performance by scenario, identify failure modes, target data collection at gaps. This is model-aware mining.

Label consistency: Ensure labelers agree. Inter-annotator disagreement indicates ambiguous cases that need clearer guidelines or expert review.

Data iteration: Treat data like code—version it, test it, iterate on it. Data bugs are as important as code bugs.

Principle: Let the model tell you what data it needs. Active learning turns labeling from a random process into a targeted optimization.

8. Labeling in the Foundation Model Era

Auto-Label Pipeline → VLM Verification → Human Review → Quality Gating → Label Store

The economics of labeling have inverted. In 2018, human labeling was the only option for high-quality annotations. In 2024, foundation models can produce many label types at 100× lower cost with comparable quality.

8.1 The New Labeling Stack

Tier 0 — Physics-Derived Labels (Free):

- Ego-motion from IMU/odometry
- Depth from LiDAR projection
- Object velocity from Doppler radar
- Lane geometry from HD map alignment

Tier 1 — Auto-Labels from Specialist Models (Pennies per frame):

- 3D object detection from offboard models (10× compute vs. onboard)
- Semantic segmentation from pretrained networks
- Tracking from multi-hypothesis trackers with global optimization
- Lane marking from HD map + learned refinement

Tier 2 — VLM-Assisted Labels (Cents per frame):

- Scene descriptions and unusual object identification
- Behavior classification (aggressive driver, distracted pedestrian)
- Anomaly flagging (debris, animals, unusual vehicles)
- Quality verification of Tier 1 outputs

Tier 3 — Human Labeling (Dollars per frame):

- Novel object classes the model has not seen
- Ambiguous cases where VLMs disagree
- Safety-critical ground truth for evaluation sets
- Disputed labels requiring expert judgment

8.2 VLM-Based Labeling Workflow

For a new segment requiring labels:

1. **Auto-label:** Run offboard detection/tracking to produce draft annotations
2. **VLM verify:** Feed frames + draft labels to a VLM, ask "Are these labels correct? What is missing?"
3. **Confidence routing:** High-confidence labels go directly to training; low-confidence routes to human review
4. **Human review:** Labelers correct VLM-flagged issues, do not start from scratch
5. **Quality gate:** Statistical checks before labels enter training pool

This hybrid approach achieves 95%+ of human quality at 10–20% of the cost.

8.3 Self-Supervised Pretext Tasks

Some "labels" do not require annotation at all:

- **Contrastive learning:** Same scene from different viewpoints should have similar embeddings
- **Temporal prediction:** Predict future frames from past frames
- **Cross-modal prediction:** Predict depth from RGB, predict semantics from LiDAR

- **Masked reconstruction:** Reconstruct masked portions of the input

These pretext tasks provide dense supervision for representation learning without any human labeling.

Principle: Humans label what models cannot. The labeling funnel should be narrow at the top—most data gets auto-labeled; humans focus on the irreducible core.

9. Synthetic Data Generation

Scenario Specification → Physics Simulation + Neural Generation → Domain Adaptation → Validation → Training Pool

Synthetic data is no longer supplementary—it is a first-class pillar of the data engine. Two paradigms coexist:

9.1 Physics-Based Simulation

Traditional simulators (CARLA, LGSVL, NVIDIA Isaac) model the world from first principles:

Sensor simulation: Ray tracing for cameras and LiDAR, wave propagation for radar. Requires accurate models of sensor noise, lens distortion, motion blur, LiDAR beam patterns.

Agent behavior: Other vehicles, pedestrians, cyclists follow behavior models ranging from rule-based (follow traffic laws) to learned (imitation from real logs) to adversarial (actively stress-test the ego).

Rare event injection: Insert scenarios that are too dangerous to encounter naturally: pedestrian darting into traffic, brake failure on lead vehicle, debris falling from truck.

Physics-based strengths: Perfect labels (you know ground truth), controllable scenarios, infinite variations, safe testing of dangerous situations.

Physics-based weaknesses: Domain gap (simulated data does not look exactly like real data), expensive to build photorealistic environments, hard to capture the full distribution of real-world weirdness.

9.2 Neural World Models

The 2023+ frontier: generative models that synthesize sensor data directly.

World models: GAIA-1 (Wayve, June 2023), GAIA-2 (2024), DriveDreamer, and others generate photorealistic driving video conditioned on:

- Initial frame(s)
- High-level actions (turn left, accelerate)
- Text descriptions ("rainy night, highway, heavy traffic")

GAIA-3 (December 2025) at 15B parameters generates coherent multi-minute driving sequences with controllable scenarios.

Neural scene reconstruction: UniSim (Waabi, CVPR 2023) takes a real driving log and creates a manipulable neural scene. You can move actors, change viewpoints, insert new objects—all while maintaining photorealism.

Diffusion-based generation: MagicDrive, DriveDiffusion, and others use diffusion models to generate driving scenes from 3D layouts, enabling precise control over object placement.

NeRF and Gaussian Splatting: Neural radiance fields and 3D Gaussian splatting enable view synthesis from sparse captures. Useful for sensor simulation and viewpoint augmentation.

Neural generation strengths: Photorealistic outputs, learns the true visual distribution, can generate scenarios that are hard to model explicitly.

Neural generation weaknesses: Labels are approximate (inferred, not ground truth), can hallucinate implausible physics, expensive training, potential for mode collapse.

9.3 Language-Guided Scenario Generation

Foundation models enable natural language scenario specification:

> "Generate a scenario where the ego vehicle is on a two-lane highway at dusk, a deer crosses from the right shoulder, and there is an oncoming vehicle in the left lane preventing lane change."

The language model translates this to simulation parameters or conditions a neural generator. This dramatically lowers the barrier to creating diverse scenarios—domain experts can describe what they want without programming.

9.4 Generation $\leftrightarrow$ Training Feedback Loop

The synthetic data loop is bidirectional:

1. **Forward:** Generate synthetic data $\rightarrow$ train model $\rightarrow$ evaluate
2. **Backward:** Identify failure modes $\rightarrow$ generate targeted scenarios $\rightarrow$ retrain

This creates a curriculum: start with easy synthetic scenarios, progressively generate harder ones as the model improves. The data generation responds to the model's current weaknesses.

9.5 Domain Gap Management

Synthetic data only helps if the domain gap is managed:

- **Domain randomization:** Vary textures, lighting, weather, sensor noise to force learning of invariant features
- **Domain adaptation:** Explicitly adapt synthetic features to real distribution (CycleGAN, adversarial training)
- **Mixed training:** Always include real data to anchor the distribution
- **Real-world validation:** Synthetic-only metrics are misleading; always validate on real benchmarks

Principle: Generate what you cannot collect. Synthetic data fills the gaps that real collection cannot reach—rare events, dangerous scenarios, future sensor configurations.

10. Training for Modern Architectures

Dataset Assembly $\rightarrow$ Curriculum Design $\rightarrow$ Multi-Task Training $\rightarrow$ Foundation Model Fine-Tuning $\rightarrow$ Validation Checkpoints

10.1 Dataset Assembly

For a training run, the data engine assembles a dataset by combining:

- **Real data:** Mined and labeled segments from the fleet
- **Synthetic data:** Generated scenarios targeting coverage gaps
- **Augmented data:** Real data with applied transformations

The ratio depends on the task and model type. A rough heuristic: 50–70% real, 20–40% synthetic, 10–20% augmented.

10.2 Curriculum Design

Modern training follows a curriculum:

1. **Pretraining phase:** Large-scale self-supervised learning on unlabeled data
2. **Foundation phase:** If using a pretrained vision backbone (CLIP, DINOv2), adapt to driving domain
3. **Task training:** Supervised training on labeled data for specific tasks
4. **Fine-tuning:** Targeted training on mined hard examples
5. **Synthetic stress-testing:** Final training with adversarial synthetic scenarios

10.3 Multi-Task Training

Modern AV models are multi-task: a shared backbone feeds detection, segmentation, prediction, and planning heads. Training considerations:

- **Task weighting:** How much gradient signal from each head?
- **Auxiliary tasks:** Self-supervised objectives as regularization
- **Gradient balancing:** Prevent dominant tasks from overwhelming others
- **Head-specific data:** Some data valuable for detection but not prediction

10.4 Foundation Model Integration

Two patterns for leveraging foundation models:

Frozen backbone + learned heads: Use a pretrained vision encoder (CLIP, DINOv2, SAM) as a frozen feature extractor. Train only the task-specific heads. Fast to train, but limited adaptation.

Full fine-tuning: Start from pretrained weights but update the entire model. Better performance but requires more data and careful learning rate scheduling to avoid catastrophic forgetting.

LoRA/adapters: Insert small trainable adapters into a frozen backbone. Compromise between the above approaches.

10.5 Training Infrastructure

Scale requirements for 2024 AV training:

- **GPU cluster:** Hundreds to thousands of GPUs (H100s or equivalent)
- **Data throughput:** Petabyte-scale storage with high-bandwidth access
- **Experiment tracking:** MLflow, Weights & Biases, or custom solutions
- **Distributed training:** PyTorch FSDP, DeepSpeed, or Megatron-style parallelism

Principle: Train on what matters. Curriculum design and task weighting are as important as model architecture.

11. Evaluation and Deployment

Holdout Sets → Scenario Slices → World Model Evaluation → Safety Gates → Shadow Mode → Deployment

11.1 Holdout Set Design

The evaluation set must be rigorous:

- **Geographic diversity:** Different cities, road types, driving cultures
- **Temporal diversity:** Different times of day, seasons, weather
- **Scenario diversity:** Coverage of ODD (Operational Design Domain) requirements
- **Human-labeled ground truth:** No auto-labels in evaluation sets

Holdout sets are refreshed periodically with newly collected data, but always maintained as strictly separate from training.

11.2 Scenario-Sliced Evaluation

Aggregate metrics hide failures. Evaluation must slice by scenario:

- **Weather:** clear, rain, fog, snow, night
- **Road type:** highway, urban, residential, intersection, construction
- **Agent types:** pedestrians, cyclists, motorcycles, trucks, emergency vehicles
- **Maneuvers:** lane change, left turn, merge, unprotected turn
- **Edge cases:** occlusion, unusual objects, aggressive actors

A model might score 95% overall while scoring 60% on "cyclist at night in rain"—which is where the safety risk lives.

11.3 World Model Evaluation

Neural world models enable a new evaluation paradigm: closed-loop testing in generated scenarios.

- **Scenario generation:** Generate thousands of scenario variations from language descriptions
- **Policy evaluation:** Run the AV policy in each scenario, measure outcomes
- **Counterfactual testing:** "What if the pedestrian had stepped out 0.5s earlier?"
- **Stress testing:** Adversarially search for failure-inducing scenarios

GAIA-3 and similar models can generate plausible multi-minute scenarios with controllable difficulty. This extends evaluation coverage far beyond collected real scenarios.

Caveat: World model evaluation has limitations. The generator may not cover all real-world edge cases. Metrics in simulation may not transfer perfectly to reality. World model evaluation complements, but does not replace, real-world testing.

11.4 Safety Gates

No model deploys without passing gates:

- **No regression:** Performance on all scenario slices must match or exceed baseline
- **Safety metrics:** TTC violations, hard braking rates, minimum distance violations
- **Coverage:** Model must have been evaluated on required scenario types
- **Human review:** Safety team sign-off on any concerning failure modes

11.5 Shadow Mode and Staged Rollout

Deployment is staged:

1. **Shadow mode:** Model runs in parallel with production, predictions logged but not used for control
2. **Limited fleet:** Deploy to small percentage of vehicles, monitor closely
3. **Gradual rollout:** Increase percentage while monitoring metrics
4. **Full deployment:** Only after stability demonstrated at scale

Principle: Evaluation drives the data engine. Failures in evaluation create the mining queries that pull new data through the system.

12. Failure Modes and Mitigations

12.1 Feedback Collapse

Auto-labeling systems trained on their own outputs converge to self-consistent nonsense. The model learns to reproduce its own errors.

Prevention: Strict separation of models that generate labels from models that train on them; human-in-the-loop validation; anchor sets with human labels.

12.2 Distribution Drift

The fleet's driving patterns change over time: new geographies, seasonal variation, changes in traffic patterns. Models trained on old data fail on new conditions.

Prevention: Continuous monitoring of input distribution; trigger retraining when drift detected; always include recent data in training.

12.3 Scenario Starvation

The mining loop converges to a narrow scenario distribution, neglecting important edge cases because they are never collected.

Prevention: Stratified sampling requirements; synthetic data to fill gaps; coverage metrics that alarm on neglected slices.

12.4 Synthetic Domain Gap

Models learn features specific to simulation: rendering artifacts, idealized physics, deterministic agent behaviors. These do not exist in reality.

Prevention: Aggressive domain randomization; real-world validation at every iteration; multiple simulation backends to avoid single-simulator overfitting.

12.5 World Model Hallucination

Neural generators can create plausible-looking but physically impossible scenarios: cars driving through walls, pedestrians teleporting, impossible lighting. Training on hallucinated data corrupts the model.

Prevention: Physics consistency checking; filtering via discriminator networks; human review of generator outputs.

12.6 Governance Failures

Without rigorous tracking, you cannot answer basic questions: which data trained this model? Who labeled this segment? What synthetic scenarios were included? This makes debugging impossible and creates regulatory risk.

Prevention: Complete provenance from collection through deployment; immutable versioning; comprehensive audit logs.

13. Open Datasets and the Research Ecosystem

The AV data engine has been dramatically accelerated by open datasets. They provide benchmarks for evaluation, starting points for research, and forcing functions for standardization.

Table 2: Major Open Datasets for Autonomous Driving

Dataset	Release	Org	Key Features
KITTI	2012	KIT/Toyota	Original benchmark, stereo + LiDAR
nuScenes	Mar 2019	Motional	Full sensor suite, 1000 scenes
Waymo Open	Aug 2019	Waymo	1000 segments, diverse geographies
Argoverse	2019	Argo AI	HD maps, tracking + forecasting
Lyft Level5	2019	Lyft	Large scale, prediction focus
nuPlan	2022	Motional	Planning, closed-loop evaluation
Waymo Motion	2021	Waymo	Motion forecasting, interactive

13.1 Major Open Datasets

13.2 Impact on Research

These datasets enabled several research breakthroughs:

- **BEVFormer** (March 2022) demonstrated transformer-based BEV perception, validated on nuScenes
- **UniAD** (2023) showed end-to-end autonomous driving, benchmarked on nuScenes
- **Standardized metrics** (NDS, mAP) enable meaningful comparison across methods
- **Reproducible research** accelerates iteration—anyone can build on published work

13.3 Limitations of Open Datasets

Open datasets are necessary but not sufficient for production:

- **Scale:** Open datasets have thousands of segments; production systems train on millions
- **Freshness:** Open datasets are static snapshots; production needs continuous updates
- **Domain specificity:** Open datasets cover general driving; production needs coverage of your specific ODD
- **Labeling quality:** Open dataset labels may not meet production safety standards

Principle: Open datasets accelerate research; proprietary data engines win in production. Use open datasets for benchmarking and initial development, but the data engine's competitive advantage comes from the closed loop on your own fleet.

14. Cross-Domain Patterns

The problems in AV data engines appear across domains. I have encountered them in healthcare AI, document intelligence, and search systems. The sensors change; the abstractions transfer.

A few specific parallels from my experience:

Healthcare claims processing → AV scenario mining. In healthcare, we mined claims data for rare disease patterns—conditions that appear in 0.01% of records but drive disproportionate cost. The technique was embedding-based similarity search with knowledge graph expansion: given one rare case, find related cases even if surface features differ. This transfers directly to AV: given one pedestrian-jaywalking-at-night scenario, find related scenarios across the fleet.

Document layout understanding → sensor fusion. Document AI requires fusing text, layout, tables, and images into unified understanding. The architecture—cross-attention between modalities with shared representations—is structurally identical to camera-LiDAR fusion. The modalities differ; the fusion pattern does not.

Clinical NLP labeling → AV auto-labeling. In healthcare, we combined billing codes, clinical notes, lab values, and imaging reports as noisy labeling functions. No single source was reliable; fusion with learned source accuracy produced usable labels. This is exactly the auto-labeling architecture for AV: HD map alignment, LiDAR-camera fusion, temporal consistency, and neural networks as weak labelers that must be combined carefully.

The abstraction insight: Data engines are domain-agnostic infrastructure. The sensor-specific components (camera decoders, LiDAR motion compensation) are implementation details. The core machinery—closed-loop learning, embedding-based mining, tiered labeling, scenario-sliced evaluation, synthetic data generation—transfers across any domain where you are learning from operational experience at scale.

Table 3: Cross-Domain Pattern Transfer

AV Problem	Adjacent Domain	Shared Technique
Rare scene discovery	Long-tail query mining (search)	Embedding similarity + knowledge graph expansion
Sensor fusion	Multimodal document AI	Cross-attention, shared representations
Auto-labeling	Weak supervision (healthcare NLP)	Labeling function fusion, learned source accuracy
Coverage gaps	Distribution shift (clinical models)	Stratified sampling, OOD detection
Data governance	HIPAA compliance (healthcare)	Immutable lineage, audit trails, access control
Difficulty scoring	Query difficulty (search ranking)	Ensemble disagreement, influence functions
Feedback collapse	Model drift (recommender systems)	Holdout anchors, pipeline separation
Synthetic data generation	Data augmentation (medical imaging)	GAN-based synthesis, physics simulation
Foundation model integration	Transfer learning (NLP)	Frozen backbones, adapter layers, fine-tuning

15. Interface Specifications

Concrete interfaces make the architecture implementable.

15.1 Segment Schema

```
Segment:
  segment_id: UUID
  vehicle_id: string
  start_timestamp_ns: int64
  end_timestamp_ns: int64
  camera_frames: list[CameraFrame]
  lidar_sweeps: list[LidarSweep]
  radar_returns: list[RadarReturn]
  vehicle_states: list[VehicleState]
  map_context: LocalMap
  calibration: CalibrationBundle
  triggers: list[TriggerEvent]
  visual_embedding: float[512]
  bev_embedding: float[256]
  language_description: string
  tags: list[ScenarioTag]
  source: REAL | SYNTHETIC | AUGMENTED
  provenance: ProvenanceRecord

SyntheticProvenance:
  generator_version: string
  scenario_spec: ScenarioSpecification
  seed: int64
  domain_randomization_params: map[string, float]
  validation_score: float
```

15.2 Mining Service

```
MiningService:
  search_similar(query_embedding, k, filters)
    -> list[SegmentMatch]
  search_by_language(query_text, k, filters)
    -> list[SegmentMatch]
  find_coverage_gaps(dataset_id, target_odd)
    -> CoverageReport
  score_segments(segment_ids, model_ids, config)
    -> list[ScoredSegment]
  get_model_data_needs(model_id) -> DataRequirements
  request_mining_job(request) -> job_id

ScoringConfig:
  novelty_weight: float = 0.3
  difficulty_weight: float = 0.3
  coverage_weight: float = 0.2
  model_gradient_weight: float = 0.2
  compute_influence: bool = false  # expensive
```

15.3 Labeling Service

```
LabelingService:
  submit_for_labeling(segment_ids, config) -> batch_id
  get_batch_status(batch_id) -> BatchStatus
  get_labels(segment_id, version) -> SegmentLabels
  run_quality_check(segment_ids) -> QualityReport
  run_vlm_verification(segment_ids, labels)
    -> VerificationReport

LabelingConfig:
  pipeline: AUTO | VLM_VERIFIED | HUMAN_REVIEW
          | FULL_MANUAL
  label_types: list[string]
  quality_threshold: float = 0.85
  vlm_model: string = "gpt-4v"
  max_latency_hours: int = 24
```

15.4 Synthetic Data Service

```
SyntheticDataService:
  generate_from_spec(scenario_spec)
    -> generation_job_id
  generate_from_language(description, count, config)
    -> generation_job_id
  generate_variations(segment_id, variation_config)
    -> generation_job_id
  validate_generation(segment_ids) -> ValidationReport
  get_generation_status(job_id) -> GenerationStatus

ScenarioSpecification:
  road_type: HIGHWAY | URBAN | RESIDENTIAL
           | INTERSECTION
  weather: list[WeatherCondition]
  time_of_day: TimeRange
  actors: list[ActorSpecification]
  ego_trajectory: TrajectorySpecification
  duration_seconds: float
  variations: int = 100
```

15.5 Evaluation Service

```
EvaluationService:
  run_eval_suite(model_id, config) -> eval_job_id
  run_world_model_eval(model_id, scenario_specs,
    config) -> eval_job_id
  get_eval_results(eval_job_id) -> EvalResults
  check_deployment_gate(model_id, baseline_id,
    target) -> DeploymentDecision
  analyze_failures(model_id, threshold)
    -> FailureAnalysis
  get_scenario_slice_metrics(eval_job_id, slice_spec)
    -> SliceMetrics

WorldModelEvalConfig:
  world_model: string = "gaia-3"
  scenarios_per_spec: int = 100
  max_duration_seconds: float = 30.0
  adversarial_search: bool = true
```

16. Future Directions: Agentic and Algebraic Data Engines

The architecture described in this document is human-orchestrated: engineers identify gaps, write queries, configure pipelines. Two emerging directions point toward more autonomous and formally rigorous systems.

16.1 Agentic Data Engines

Current data engines react to human-identified problems. An *agentic* data engine autonomously diagnoses, hypothesizes, and acts. When pedestrian detection degrades in rain, the agent notices, investigates possible causes (insufficient data, label noise, synthetic domain gap), and executes corrective actions—commissioning fleet collection, triggering synthetic generation, re-labeling suspicious segments—without human intervention.

LLM agents with tool use make this feasible. The data engine's APIs (mining, labeling, generation, training) become the agent's action space. The agent plans, acts, observes results, and iterates. Human engineers shift from operators to supervisors.

16.2 Algebraic Types for Data Engines

Current systems represent scenarios as strings, pipelines as implicit code, provenance as best-effort metadata. *Algebraic types* formalize these as composable, verifiable structures:

- **Scenario algebra:** Scenarios as product types (Weather $\times$ RoadType $\times$ ActorConfig), enabling formal coverage analysis and type-safe generation
- **Query algebra:** Mining queries as typed expressions with algebraic laws for optimization
- **Provenance algebra:** Data lineage as typed values—every example traceable through transformations to source
- **Pipeline types:** Dataflow stages as typed functions, ensuring invalid configurations (training on uncalibrated data) are caught at compile time

The combination is powerful: agents operate over algebraic structures, constrained by type systems to only take valid actions. Formal methods meet LLM agents.

These directions represent a significant expansion of scope and will be addressed in a companion document.

17. Conclusion

The data engine architecture has evolved dramatically since 2017. What started as "collect real data, have humans label it" has become a sophisticated dual-loop system integrating real collection, synthetic generation, foundation model labeling, model-aware mining, and world model evaluation.

The key architectural decisions for 2024+:

Dual-loop operation. Real data grounds you in reality; synthetic data reaches the long tail. Every modern system operates both loops.

Model-aware mining. Different architectures need different data. Mining must understand what the current model actually needs to improve.

Foundation model labeling. VLMs and LLMs have inverted the economics. Humans label what models cannot—which is a shrinking set.

Synthetic data as first-class citizen. 30–50% of training data may be synthetic, generated by physics simulators, neural world models, and diffusion-based generators.

World model evaluation. Neural scene generation enables closed-loop testing at scale. Counterfactual scenarios, adversarial search, and stress testing extend evaluation far beyond collected data.

Complete provenance. From deployed model back to training data (real and synthetic) back to generation parameters and collection events. Debugging and governance require full traceability.

The patterns from adjacent domains transfer directly. I have built these systems for healthcare and document intelligence; the sensors change but the abstractions do not. Embedding-based mining, tiered labeling, scenario-sliced evaluation, feedback loop prevention—these are domain-agnostic techniques.

Build the data engine first. The models will follow.

References

Industry Sources

- Tesla AI Day 2021 (August 19, 2021): Data Engine, auto-labeling, 1000+ person labeling organization
- Tesla AI Day 2022 (September 30, 2022): FSD scaling, Dojo supercomputer architecture
- Waymo: Offboard 3D Object Detection (CVPR 2021), Safety Data Hub

Open Datasets

- nuScenes (Motional, March 2019): First large-scale multimodal AV dataset
- Waymo Open Dataset (August 2019): High-quality perception and motion forecasting
- nuPlan (Motional, 2022): Planning benchmark with closed-loop evaluation

Key Papers

- CARLA Simulator (CoRL, November 2017): Open-source driving simulator
- Domain Randomization (Tobin et al., March 2017): Sim-to-real transfer technique
- BEVFormer (ECCV 2022): Camera-only BEV perception with transformers
- UniSim (Waabi, CVPR 2023): Neural closed-loop sensor simulation
- GAIA-1 (Wayve, June 2023): World model for autonomous driving
- CLIP (OpenAI, January 2021): Vision-language pretraining

Standards

- ISO 26262 (Functional Safety)
- ISO 21448 (SOTIF — Safety of the Intended Functionality)
- UL 4600 (Autonomous Vehicle Safety)
- PEGASUS Project: Scenario-based Testing Framework