Data Selection II: Engineering Data-Efficient ML Systems

Synthetic Data, Conditional Diffusion, and End-to-End Systems Trade-offs

Quick Recap: What Did Lecture 1 Optimize?

- **Static pruning:** Which existing examples should we keep?
- **Dynamic selection:** Which examples or labels are useful *now*?
- **Synthetic data:** What if the examples we need *do not exist*?

Smaller datasets and more useful datasets are not the same objective.

Today's Question: When Do We Need More Data?

Smart doorbell: The model detects pedestrians in residential driveways.

Scene	Real training examples	Recall on real evaluation data
Daylight pedestrian	20,000	97%
Nighttime pedestrian	1,000	83%
Rainy nighttime pedestrian	100	62%
Partially occluded pedestrian	500	75%

Hypothetical numbers for classroom discussion. **Q:** We have two GPUs and an annotation budget. Do we prune, augment, generate, or improve the data pipeline?

First diagnose the bottleneck.

Two Different Optimization Problems

Case A: Redundant Data

We have a large labeled dataset.

$D_{\text{subset}} \subset D_{\text{full}}$

$T_{\text{select}} + T_{\text{train}}(D_{\text{subset}})$
$< T_{\text{train}}(D_{\text{full}})$

Case B: Missing Data

Required examples are scarce.

$D_{\text{train}} = D_{\text{real}} \cup D_{\text{synth}}$

$\min C_{\text{total}}$
s.t. $Q_{\text{real}} \geq Q_{\text{target}}$

Discussion: Why is "generate + train < full-data training" the wrong default comparison in Case B?

Augmentation vs. Generative Synthesis

	Augmentation	**Generative synthesis**
Input	Existing labeled image	Prompt, scene parameters, image or structural condition
Transformation	Modify existing input	Create a plausible new example
Typical methods	Crop, jitter, MixUp, CutMix, RandAugment	Simulation, GANs, diffusion models
Labels	Usually inherited/adjusted	Must be generated or validated
Systems cost	Often low, on-the-fly	Generation, screening, storage, training
Major risk	Transformation breaks the label	Synthetic–real domain gap

Q: Can brightness jitter create a physically correct rainy nighttime scene with reflections and occlusions?

Key idea: Transformation-based augmentation is often cheaper; generative synthesis may fill gaps it cannot.

Concrete Baseline: MixUp

Given two labeled examples $(\mathbf{x}_i, \mathbf{y}_i)$ and $(\mathbf{x}_j, \mathbf{y}_j)$,

$$\boxed{\tilde{\mathbf{x}} = \lambda \mathbf{x}_i + (1-\lambda)\mathbf{x}_j, \qquad \tilde{\mathbf{y}} = \lambda \mathbf{y}_i + (1-\lambda)\mathbf{y}_j}$$

- $\lambda \in [0, 1]$, often sampled from a Beta distribution.
- Interpolate *both* images and class-label vectors.

Worked example: Image A = pedestrian, image B = empty driveway, $\lambda = 0.7$.

$$\tilde{\mathbf{y}} = 0.7(1, 0) + 0.3(0, 1) = (0.7, 0.3).$$

The model is trained with a *soft target*, not a hard label "person."

Zhang et al., *mixup: Beyond Empirical Risk Minimization*, ICLR 2018.

Why Not Just Use MixUp or RandAugment?

- You have 100 real rainy-night frames and 20,000 daylight frames.
- Augmentations can vary brightness, crop, noise and geometric appearance.
- But the training set may still lack realistic scene structure:
  - headlight reflections on wet surfaces;
  - an umbrella partly blocking the pedestrian;
  - fog, depth-dependent rain, and multiple light sources.

Can we **generate a new scene under specified conditions**?

Synthetic Generation: What Must an Algorithm Do?

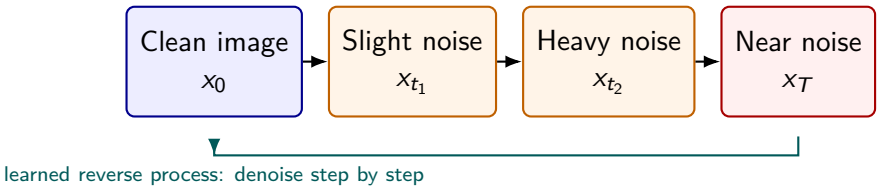
We want an image generator that produces:

"A pedestrian crossing a residential driveway at night in heavy rain."

1. Learn a distribution of plausible images.
2. Support **conditioning** on desired scene properties.
3. Produce diverse samples, not just duplicates.
4. Retain or verify correct downstream labels.

Our detailed method: Conditional Latent Diffusion

Diffusion: Learn to Reverse Noise



- **Forward process:** progressively corrupt a real training image.
- **Learning task:** predict the added noise.
- **Generation:** start with random noise and repeatedly denoise.

Ho, Jain and Abbeel, *Denoising Diffusion Probabilistic Models*, NeurIPS 2020.

Forward Diffusion: One Equation

At a randomly chosen noise level t, construct a noisy image:

$$\boxed{x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I)}$$

- x_0: clean training image.
- $\bar{\alpha}_t$: fraction of original signal power remaining.
- ϵ: sampled Gaussian noise.
- x_t: noisy training input at noise level t.

Q: When $\bar{\alpha}_t \approx 1$, what does x_t look like? What about $\bar{\alpha}_t \approx 0$?

Worked Example: Construct One Noisy Pixel

For a single normalized value let:

$$x_0 = 1, \qquad \bar{\alpha}_t = 0.64, \qquad \epsilon = -1.$$

Then:

$$x_t = \sqrt{0.64}(1) + \sqrt{0.36}(-1) = 0.8 - 0.6 = \boxed{0.2}.$$

Q: Does the model need a human label such as "pedestrian" to construct this training pair?

No. It knows the injected noise because the training pipeline sampled it.

Train a Neural Network to Predict the Noise

Input to the network: noisy sample x_t and noise level t.

$$\hat{\epsilon} = \epsilon_\theta(x_t, t).$$

We know the true noise ϵ because we sampled it.

$$\boxed{\mathcal{L}_{\text{noise}} = \mathbb{E}_{x_0,t,\epsilon}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right]}$$

1. Sample clean image x_0 and noise level t.
2. Sample Gaussian ϵ and construct x_t.
3. Predict $\hat{\epsilon}$ using the network.
4. Minimize squared noise-prediction error; repeat.

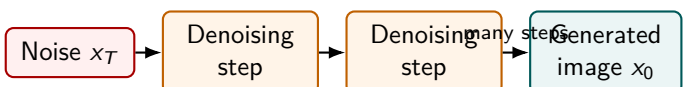
How Does Image Generation Work?

- During **training**, start with an image and add noise.
- During **sampling**, start with random noise:

$$x_T \sim \mathcal{N}(0, I).$$

- Apply a numerical reverse-process sampler repeatedly:

$$x_T \to x_{T-1} \to \cdots \to x_0.$$



Key idea: Unlike ordinary augmentation, generating one example requires repeated network evaluations.

Systems Insight: Why Not Diffuse Directly in Pixels?

An RGB image of size 512×512 contains:

$$512 \times 512 \times 3 = 786{,}432 \text{ scalar values.}$$

A pretrained autoencoder can compress it into a smaller *latent* array. For illustration, suppose the latent shape is $64 \times 64 \times 4$:

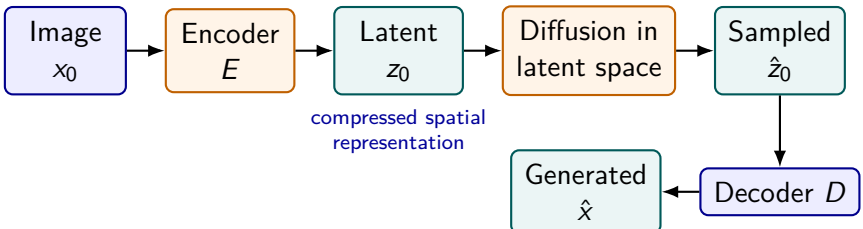
$$64 \times 64 \times 4 = 16{,}384 \text{ values.}$$

$$\frac{786{,}432}{16{,}384} = \boxed{48\times \text{ fewer scalar values}}.$$

Warning: This $48\times$ reduction in array size is *not* a claim of $48\times$ faster generation.

Rombach et al., *High-Resolution Image Synthesis with Latent Diffusion Models*, CVPR 2022.

Latent Diffusion: Architecture



- **Training:** encode real images; learn to denoise their noisy latents.
- **Generation:** denoise random latent noise; decode to an image.

Rombach et al., CVPR 2022. Vector diagram redrawn for teaching.

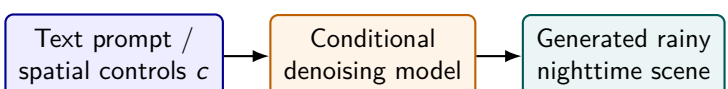
Conditional Diffusion: Ask for the Missing Scene

Unconditional generation samples plausible images. We want *specific* images.

$$\boxed{\hat{\epsilon} = \epsilon_\theta(z_t, t, c)}$$

Condition c may include:

- Text: "pedestrian at night, heavy rain, driveway".
- Class or scene metadata.
- A source image, mask, depth map or spatial structure.

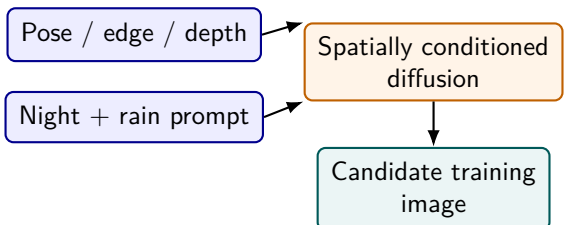


ControlNet: Text Prompt Is Not Enough

Problem: A prompt may create a pedestrian, but not in the requested location.

Additional structure:

- Human pose/skeleton.
- Edge map.
- Depth map.
- Segmentation layout.

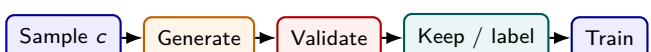


Key idea: Conditioning improves controllability, but labels and geometry still need verification.

Zhang, Rao and Agrawala, *Adding Conditional Control to Text-to-Image Diffusion Models*, ICCV 2023.

Algorithm: Build a Synthetic Nighttime Dataset

1. Specify missing coverage: rain intensity, lighting, pose, occlusion.
2. Sample scene conditions c_i (possibly with spatial controls).
3. Generate image $\tilde{x}_i \sim p_\theta(x \mid c_i)$ using a pretrained generator.
4. Validate that the desired object and scene are present.
5. Validate or repair the task label $\tilde{y}_i$.
6. Add accepted $(\tilde{x}_i, \tilde{y}_i)$ to a real/synthetic training mix.
7. Evaluate exclusively on a **held-out real test set**.



Why Visual Realism Is Not Enough: The Domain Gap

Synthetic distribution

$P_{\text{synth}}(x, y)$

May contain:

- Unrealistic reflections.
- Repeated textures or styles.
- Missing camera/sensor artifacts.
- Incorrectly placed objects.

Deployment distribution

$P_{\text{real}}(x, y)$

Includes:

- Real nighttime lighting.
- Motion blur and compression.
- Sensor-specific artifacts.
- True correlations among events.

$$P_{\text{synth}}(x, y) \neq P_{\text{real}}(x, y) \implies \text{synthetic success} \not\Rightarrow \text{real success.}$$

Worked Example: The Cost of Accepted Examples

We need **6,000 valid synthetic nighttime images**.

- A pretrained generator produces one candidate in 0.8 GPU-seconds.
- Only 60% of generated candidates pass quality/label checks.

How many images must we generate?

$$N_{\text{gen}} = \frac{6{,}000}{0.60} = \boxed{10{,}000}.$$

Generation GPU-time:

$$T_{\text{gen}} = 10{,}000(0.8\,\text{s}) = 8{,}000\,\text{s} \approx \boxed{2.22\ \text{GPU-hours}}.$$

Not included: checking labels, storage, retries, data movement, downstream training.

Which Synthetic Strategy Is Actually Better?

Experiment	Training dataset	Question answered
A: Real only	Real frames	Baseline
B: Real + RandAugment	Real with cheap transformations	Does cheap augmentation suffice?
C: Real + diffusion	Real and screened synthetic	Does generation add coverage?
D: Real + both	Real + augmentation + diffusion	Are benefits complementary?

Hold constant: model, test set, evaluation slices, random seeds, and a stated training-compute protocol.
Measure: rainy-night recall, false positives, generated-label accuracy, end-to-end cost.

Synthesis: What Did We Learn About Generation?

- Diffusion models learn to reverse a noise process.
- Latent diffusion performs denoising in a compressed representation.
- Conditioning helps generate targeted, missing scenarios.
- Acceptance/label validation are part of the *algorithmic pipeline*.
- The objective is **deployment quality per total resource cost**, not necessarily fewer examples.

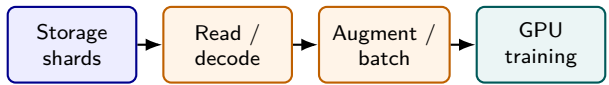
A smarter dataset is useful only if the training system can process it efficiently.

10-minute break

Next: data pipelines, I/O bottlenecks, selection overhead, and amortization.

Why Does a Smaller Dataset Not Guarantee Faster Training?

Think beyond the neural network:



- Every stage must feed the next one quickly enough.
- Training can be compute-bound, I/O-bound, or preprocessing-bound.
- Selection can reduce operations but *change access patterns*.

Q: If the GPU can process 800 images/s but the input pipeline delivers 300 images/s, what is the sustained training throughput?

The Pipeline Bottleneck: Throughput

In a steady-state pipeline with overlap:

$$\boxed{R_{\text{effective}} \approx \min(R_{\text{input}}, R_{\text{GPU}})}$$

Suppose:

$$R_{\text{input}} = 300 \text{ images/s}, \qquad R_{\text{GPU}} = 800 \text{ images/s}.$$

$$R_{\text{effective}} = \boxed{300 \text{ images/s}}$$

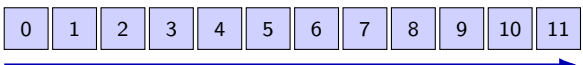
Approximate GPU busy fraction, if it waits only for input:

$$\frac{300}{800} = \boxed{37.5\%}.$$

Key idea: A more efficient model does not accelerate a job whose bottleneck is elsewhere.

Selection Can Turn Sequential I/O into Random I/O

Original dataset: stored sequentially in large shards.



Sequential read

Selected dataset: examples may be scattered across the original shards.



Index-driven reads

- Fewer samples can mean smaller reads, more seeks, and less prefetch benefit.
- Repack selected examples into contiguous shards, or use shard-aware sampling.

Worked Example: Pruning, but Data Loading Gets Slower

Suppose both jobs run for 10 epochs and reach comparable quality.

	Full dataset	Pruned subset
Examples per epoch	1,000,000	250,000
Sustained throughput	1,000/s	200/s
Time per epoch	1,000 s	1,250 s

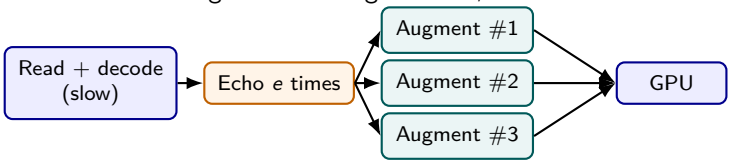
$$T_{\text{full}} = 10 \times \frac{1{,}000{,}000}{1{,}000} = 10{,}000 \text{ s}$$

$$T_{\text{pruned}} = 10 \times \frac{250{,}000}{200} = 12{,}500 \text{ s}$$

75% fewer examples, but 25% more training time.

Data Echoing: Reuse After Expensive Input Stages

Problem: Reading and decoding are slow; the accelerator waits.



- Echo *before* randomized augmentation to produce multiple views.
- Echo *after* augmentation may repeat identical tensors.

Choi et al., *Faster Neural Network Training with Data Echoing*, 2019.

Worked Example: Does Echoing Help This Pipeline?

Assume input reads/decodes **300 unique images/s**; GPU accepts up to **800 training views/s**.

$$R_{\text{views}}(e) \approx \min(300e, 800).$$

Echo factor e	Available views/s	GPU capacity	Bottleneck
1	300	800	Input
2	600	800	Input
3	900	800	GPU

Q: Is $e = 3$ automatically the best training strategy?

No! More transformed views per second may not mean faster *learning*.

Selection Has an Upfront Cost

Example: k-center / EL2N selection needs work before subset training.

$$T_{\text{selected}} = T_{\text{score}} + T_{\text{choose}} + T_{\text{train,subset}} + T_{\text{I/O overhead}}$$

- Embedding computation and similarity search (k-center).
- Proxy training/inference (EL2N).
- Repeated rescoring (dynamic selection).
- Repacking selected samples into efficient storage shards.

Q: Can a selection method that halves training work still make the whole job slower?

Yes, if selection and I/O overhead exceed the saved training time.

Numerical Break-Even Example

Measured training times:

Operation	Time
Train full dataset (one run)	16 hours
Score and select subset (one-time)	6 hours
Train selected dataset (one run)	12 hours

One run:

$$T_{\text{full}} = 16\,\text{h}, \qquad T_{\text{selected}} = 6 + 12 = 18\,\text{h}.$$

Four runs reusing one subset:

$$T_{\text{full}} = 4(16) = 64\,\text{h}, \qquad T_{\text{selected}} = 6 + 4(12) = 54\,\text{h}.$$

Amortization: How Many Reuses Pay for Selection?

Let:

- C_s: one-time selection cost.
- C_f: full training cost per run.
- C_r: reduced training cost per run.
- n: number of runs reusing a valid selection.

$$C_s + nC_r < nC_f \quad \Longleftrightarrow \quad \boxed{n > \frac{C_s}{C_f - C_r}}$$

For $C_s = 6$, $C_f = 16$, $C_r = 12$ hours:

$$n > \frac{6}{16 - 12} = \boxed{1.5}.$$

Answer: Starting at the **second** run, selection saves compute-time.

The Cost Comparison for Generation Is Different

Option A: Collect and label missing rainy-nighttime scenes.

$$C_A = C_{\text{collection}} + C_{\text{labeling}} + C_{\text{training}}.$$

Option B: Use a pretrained diffusion generator.

$$C_B = C_{\text{generation}} + C_{\text{validation}} + C_{\text{training}}.$$

The correct comparison:

$$\min\{C_A, C_B, \text{other feasible strategies}\} \quad \text{s.t.} \quad Q_{\text{real}} \geq Q_{\text{target}}.$$

Warning: A generator's *pretraining* cost is not zero, even if the team licenses a pretrained model. Account for direct costs and explain which costs are amortized or externalized.

Evaluate the ML System, Not Just the Dataset

Dimension	What to measure	Why it matters
Predictive quality	Overall accuracy; rainy-night recall; false positives	Rare cases may disappear inside averages
Statistical efficiency	Accuracy vs training examples / optimizer steps	Compare learning progress
Compute efficiency	GPU-hours; FLOPs; total wall-clock time	Selection/generation overhead counts
Data quality	Synthetic acceptance; label error; scene diversity	Generator output is not guaranteed useful
Systems behavior	I/O throughput; GPU idle time; cache behavior	Fewer examples may run slower

Experimental rule: Compare methods at fixed target quality, on held-out *real deployment* examples.

Capstone: Three Engineers, One Budget

Smart doorbell model: strong daylight performance, weak rainy-night detection.

Engineer A: Prune redundant daylight images using k-center.

Engineer B: Generate screened rainy-night images with conditional diffusion.

Engineer C: Optimize data loading, caching, and GPU utilization.

Discuss in pairs:

1. Which intervention should come first?
2. What one measurement would determine whether that choice is right?
3. Can two or three interventions be combined without extra failure modes?

Capstone Results: Which Model Do We Deploy?

Hypothetical pilot outcomes; target rainy-night recall $\geq 90\%$.

System	Overall accuracy	Rainy-night recall	Total cost
A: Pruning only	96%	65%	8 GPU-hrs
B: Synthetic + real	95.5%	91%	19 GPU-hrs
C: Better I/O only	96%	75%	10 GPU-hrs

Q: Which alternatives meet our deployment constraint?

Only B, as specified here. Cheap training is not useful if deployment requirements are unmet.

Takeaways

1. **Selection and synthesis solve different data problems.**
2. **Conditional latent diffusion** provides a concrete way to generate targeted scenarios, at nontrivial generation and validation cost.
3. **Quality matters:** a plausible image is not necessarily a correctly labeled or useful training example.
4. **Hardware matters:** random-access I/O, augmentation and GPU starvation can dominate runtime.
5. **Measure end to end:** account for selection/generation costs, amortization, and target deployment quality.

Reduce wasted work; create missing information; evaluate the whole system.

Further Reading and Source Alignment

- MLSys Book, Vol. 1: Data Selection
  Synthetic Data Generation; Selection Engineering; Cost Modeling; Measurement.
- Ho, Jain, Abbeel. *Denoising Diffusion Probabilistic Models* (NeurIPS 2020).
- Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models* (CVPR 2022).
- Zhang, Rao, Agrawala. *Adding Conditional Control to Text-to-Image Diffusion Models* (ICCV 2023).
- Choi et al. *Faster Neural Network Training with Data Echoing* (2019).
- Zhang et al. *mixup: Beyond Empirical Risk Minimization* (ICLR 2018).