Data Augmentation and Synthetic Data

DSCC/LING 251/451: Machine Learning with Limited Data

C.M. Downey

Spring 2026

Roadmap

- The augmentation **spectrum**: from perturbed real data to pure synthesis
- Classical augmentation as **invariance design** (vision, speech, text)
- **Back-translation**: the bridge case in NLP — and why it shouldn't work but does
- Synthetic data: **simulators** and **generative models**
- **LLM-generated data**: self-instruct, distillation, and model collapse
- The throughline: where the supervision signal actually lives

Augmentation in this course

The whole course is about what to do without enough labeled data

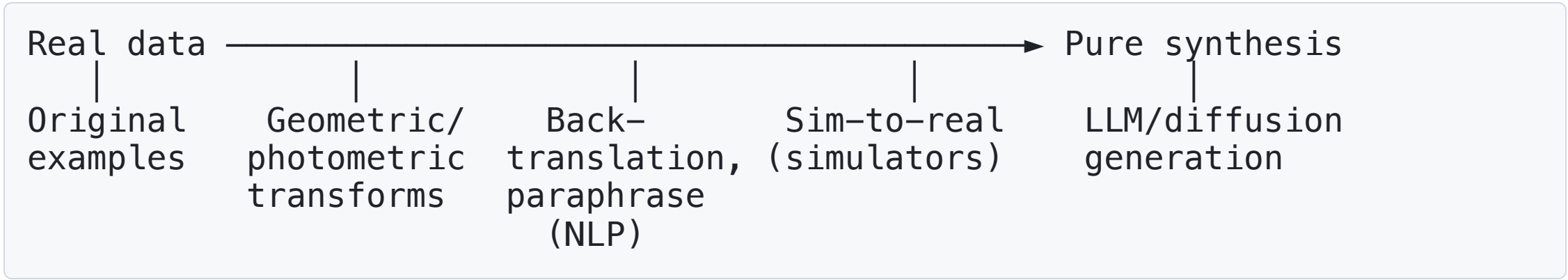
Augmentation is the most direct response: **manufacture more data**

Two flavors of "more":

1. **More variation around what you have** — perturb existing examples to make the model robust to nuisance factors
2. **Genuinely new examples** — generate data that didn't exist, hoping it covers parts of the input space the original missed

These are different things and they fail in different ways

The augmentation spectrum



- **Left end**: same example, perturbed — high fidelity to the real distribution
- **Right end**: a new example, generated from scratch — maximum flexibility, minimum guarantee that it lives on the real data manifold
- The interesting questions are in the **middle**: where you've moved far enough that label-preservation isn't obvious, but you're not yet synthesizing

Every augmentation is an inductive bias

Choosing an augmentation = choosing an **invariance**:

Idea: the model should produce the same output if I apply f to the input

Every augmentation is a **hypothesis about which features matter and which don't**

Inappropriate invariances hurt you:

- Rotation of digits → "6", "9"
- Color-jittering medical images → erases diagnostic information
- Synonym replacement on NLI → flips entailment direction

Callback to **Lecture 3**: hand-designing augmentations is hand-designing inductive bias

Classical Augmentation as Invariance Design

Vision augmentations and their invariances

Augmentation	Invariance hypothesis	When it breaks
Horizontal flip	Left/right is irrelevant	Text, digits, handed objects
Random crop	Object can be anywhere	Tasks where global layout matters
Color jitter	True color is irrelevant	Medical, plant disease, anything color-diagnostic
Rotation	Orientation is irrelevant	Aerial vs. natural scenes
Gaussian noise	Sensor noise shouldn't matter	Already-noisy data
Cutout / random erasing	No single region is essential	Almost always OK

An augmentation strategy is a **claim about the task** — choose them by asking "what should this model ignore?", not by copying defaults.

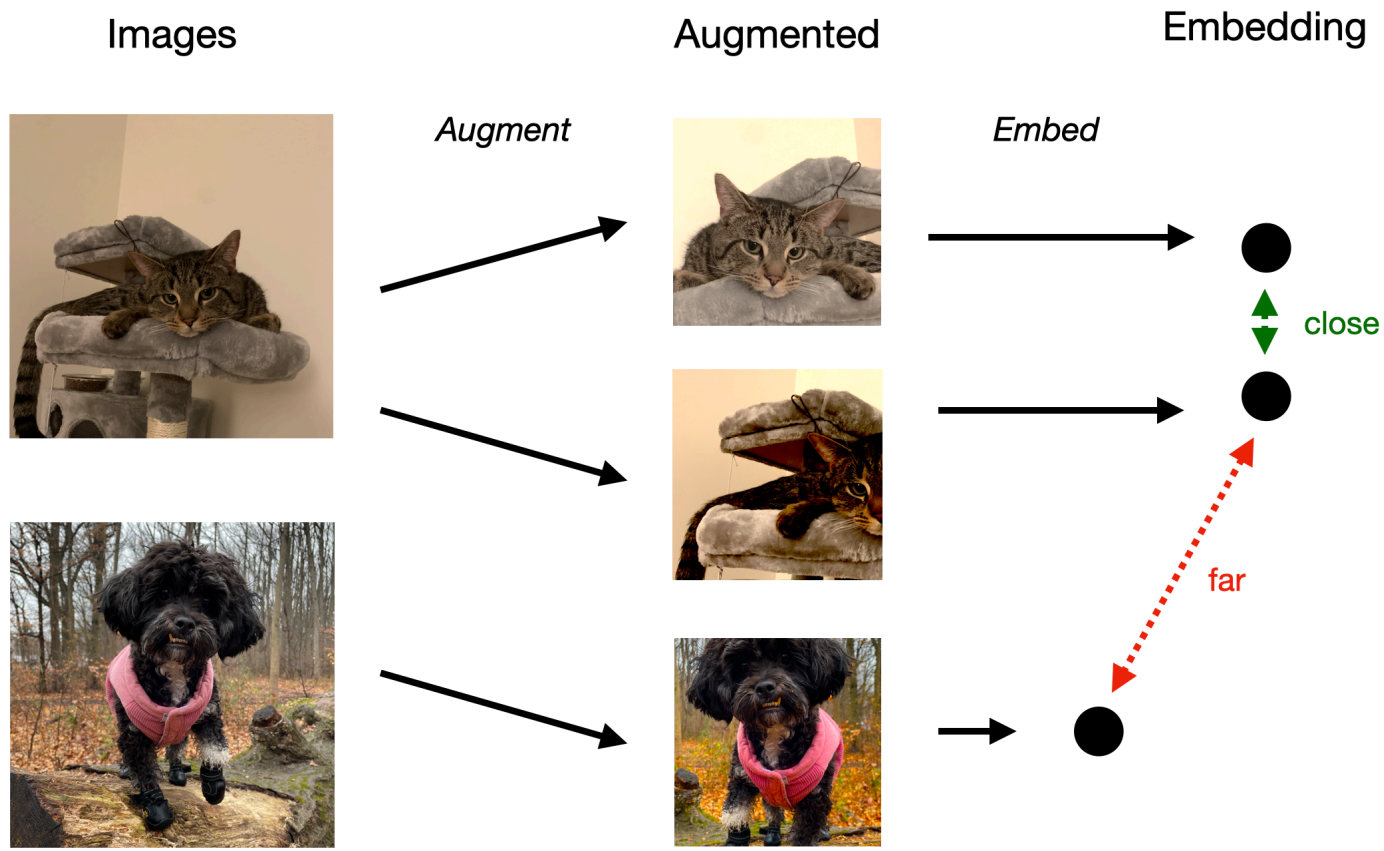
SimCLR (Lecture 7)

In contrastive self-supervised learning, the augmentation pipeline isn't just regularization — it **defines the learning signal**

- Two views of the same image are **positives**; everything else is a **negative**
- The augmentation set literally defines the equivalence class the model learns

The authors studied which augmentations actually drive contrastive learning:

- **Strong color distortion + random crop** is the magic combination
- Drop either one and performance collapses
- Choice of augmentations matters more than almost any architectural decision


Images
Augment
Augmented
Embed
Embedding
close
far

(a) Original (b) Crop and resize (c) Crop, resize (and flip) (d) Color distort. (drop) (e) Color distort. (jitter)

(f) Rotate $\{90°, 180°, 270°\}$ (g) Cutout (h) Gaussian noise (i) Gaussian blur (j) Sobel filtering

Figure 4. Illustrations of the studied data augmentation operators. Each augmentation can transform data stochastically with some internal parameters (e.g. rotation degree, noise level). Note that we *only* test these operators in ablation, the *augmentation policy used to train our models* only includes *random crop (with flip and resize)*, *color distortion*, and *Gaussian blur*. (Original image cc-by: Von.grzanka)

Speech: SpecAugment

Operates on the **spectrogram** (time × frequency), not the waveform

1. **Time warping** — speech rate variation
2. **Frequency masking** — robustness to a missing band (cheap mic, hum)
3. **Time masking** — robustness to brief dropouts (cough, packet loss)

Each operation encodes a **specific known failure mode** of ASR

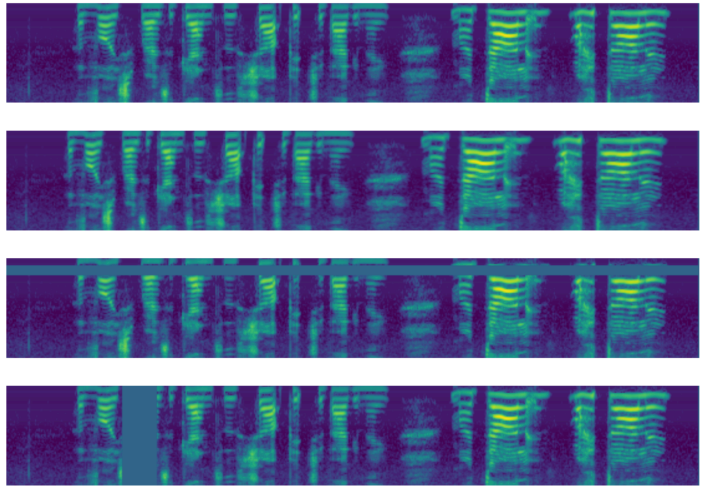


Figure 1: *Augmentations applied to the base input, given at the top. From top to bottom, the figures depict the log mel spectrogram of the base input with no augmentation, time warp, frequency masking and time masking applied.*

SpecAugment: the lesson

Each augmentation is targeted at a **known failure mode**

- Replaced ad-hoc acoustic-model regularization tricks across the field
- Now standard in ASR fine-tuning pipelines (e.g., wav2vec 2.0 fine-tuning applies a modified SpecAugment)
- Generic augmentations would not have produced these gains

Pedagogical takeaway: augmentation design rewards **domain knowledge**

If you don't know what your model should be invariant to, you don't know what to augment over

Why it's hard to augment text

Images live in a continuous space. Small pixel perturbations are still "the same image."

Text lives in a discrete combinatorial space. Small token perturbations can flip meaning entirely:

`I do not love this movie` → delete one token → `I do love this movie`

There is no analogue of "small Gaussian noise" for text that reliably preserves meaning

"Label-preserving" is a **much stronger requirement** in language because semantics are brittle

EDA: the cautionary tale

Wei & Zou (2019): **Easy Data Augmentation**

Four operations:

- **Synonym replacement** (random word → WordNet synonym)
- **Random insertion** (random synonym at random position)
- **Random swap** (swap two random words)
- **Random deletion** (drop random words)

Where it works: short, robust text classification (sentiment on movie reviews)

Where it fails: NLI, QA, syntactic tasks — anything where word order or function words matter

The lesson: **techniques that work on one NLP task don't generalize across NLP tasks**

Mixup: augmenting in a different space

Idea: don't perturb individual examples — **interpolate** between pairs

Take two examples (x_1, y_1) and (x_2, y_2) and train on:

$$\tilde{x} = \lambda x_1 + (1 - \lambda) x_2, \quad \tilde{y} = \lambda y_1 + (1 - \lambda) y_2$$

The model is asked to predict **linearly between training points** — a strong smoothness prior

- **CutMix, Manifold Mixup**: variants in pixel space and embedding space
- Almost free to implement, often improves generalization
- The augmentation doesn't have to live in input space

Back-Translation

The setup

You want to train a translation model from **language X to language Y**

You have:

- A **small** amount of *parallel* data: pairs (x, y)
- A **lot** of *monolingual* data in Y (target language)
- Standard training only uses parallel data — the monolingual data is **wasted**

Can we use the monolingual Y data somehow?

Sennrich, Haddow, & Birch (2016)

The **back-translation** procedure:

1. Train an initial model in the **reverse direction**: $Y \rightarrow X$
2. Use it to translate monolingual Y into synthetic X'
3. You now have synthetic pairs (x', y) where y **is real** and x' **is machine-generated**
4. Train your forward $X \rightarrow Y$ model on the union of real parallel data + these synthetic pairs

Why this is non-obvious

- The synthetic X' is *worse* than real X — it's machine output, full of errors
- Yet adding lots of (x', y) pairs **improves** the $X \to Y$ model — often by several BLEU points
- How does this actually work? What does the model learn from?

The key insight

The model is always trained to **produce real data**

You're not asking the model to learn how to produce noisy output

You're asking it to learn how to produce **correct, fluent** y from *some* input

- The **decoder** has to be fluent → real y teaches that
- The **encoder** just has to be robust to noisy inputs → which is also useful at test time when humans write messy source sentences

Effectively a regularizer for the encoder *and* a fluency boost for the decoder, simultaneously

Framing 1: conditional language modeling

A translation model is $p(y \mid x)$

Training on real y teaches the model what fluent y looks like, conditioned on *something*

Even with noisy x, the loss on y is well-defined and pushes the decoder toward correct y distributions

It's "language modeling with a noisy prompt"

Framing 2: consistency regularization

Callback to **semi-supervised learning (Lecture 8)**

The model should produce the same y whether the source is real x or back-translated x'

This is **exactly** the consistency regularization we saw with FixMatch — same idea, applied to sequence-to-sequence

Back-translation **pre-dates** and is a special case of the consistency-regularization view of semi-supervised learning

Framing 3: domain coverage

Real parallel data is **biased**:

- Usually from specific domains: news, EU parliament, software localization

Monolingual Y is much **broader**:

- Web text, books, social media, dialogue

Back-translation lets the model see **decoder-side coverage** that no parallel corpus would provide

The model learns to produce fluent Y for topics that never appeared in real parallel data

Edunov et al. (2018): the empirical surprise

- Sennrich's original used **beam search** to generate the synthetic source
- Edunov et al. asked: what if we use **sampling** instead?
- Result: **sampling-based back-translation works better** — sometimes substantially

Why?

- Beam search → high-likelihood, low-diversity outputs. Synthetic sources all look similar.
- Sampling → more varied, noisier sources that cover more of the input space the encoder will see at test time

Iterative back-translation

Sennrich's procedure is one pass. But there's no reason to stop:

1. Train reverse model $Y \to X$, generate synthetic (x', y)
2. Train forward model $X \to Y$ on real + synthetic
3. Now **use the improved forward model** to retrain the reverse model
4. Generate better synthetic data, retrain forward again
5. Repeat

Each iteration improves **both** models — the synthetic data gets better because the model generating it gets better

This is **self-training with a feedback loop**: exactly the kind of iterative refinement that works when you have a good anchor (real monolingual data on both sides)

The lesson: diversity > quality

For augmentation, **diversity often beats quality**

Optimal-looking synthetic data risks **collapsing to the model's mode** and teaching the next model only what it already knows

This same lesson recurs in:

- LLM-generated training data (next section)
- Sampling temperatures for synthetic CoT generation
- Domain randomization in robotics

When you're using a model to generate training data, you usually want the synthetic data to be **diverse, not optimal**

Generalizing the back-translation pattern

Use a model to generate **inputs paired with real outputs**

Same idea appears in:

- **Question generation** for QA: generate questions from passages, train QA on the synthetic questions paired with real answers
- **TTS for ASR**: synthesize audio from real transcripts → train ASR on (synthetic audio, real transcript)
- **Distillation**: the teacher generates labels for unlabeled inputs

The unifying principle: **the supervision side that you care about should be real; the side you can tolerate noise on is the one you generate**

Synthetic Data

Generation From Scratch

Simulators and sim-to-real

The original "synthetic data" story is from **robotics and self-driving**

- Build a physics simulator (CARLA, MuJoCo, Habitat)
- Generate millions of (state, action, reward) trajectories essentially for free
- Train a policy in simulation, deploy in the real world

The **sim-to-real gap**: simulators are simplified, the real world isn't

Domain randomization

Tobin et al. (2017): **randomize lighting, textures, friction, masses** across simulated episodes

Goal: make the real world "just one more variation" the policy has already seen

Domain randomization is itself a form of augmentation — you're **augmenting the simulator**, not the data

The trade-off:

- **Simulators** → unlimited data, known bias
- **Real data** → no bias, limited quantity
- **Domain randomization** pushes the simulator toward the real distribution
- **Domain adaptation** (Lecture 11) pulls the model toward generalizing across the gap

Generative models for augmentation

Use a GAN or diffusion model to generate new training images

Tempting because diffusion models are stunningly good at producing realistic images

He et al. (2023): "Is synthetic data from generative models ready for image recognition?"

- Synthetic-only training is **much worse** than real-only training
- Synthetic + real **beats** real-only — but only when synthetic is generated *conditionally*, with class labels and prompts that diversify coverage
- Gains are **modest** (1-3% on ImageNet variants)

What is the generator actually adding?

If a generator G was trained on dataset D, samples from G are (roughly) samples from $\hat{p}_D$ — the empirical distribution

Training a new model on samples from G is closer to training on a **smoothed version of** D than on genuinely new data

You **haven't added information.** You've added **inductive bias** — the smoothing structure of G.

Sometimes this is what you want. Sometimes the generator's biases get amplified instead.

Data Processing Inequality: synthetic data cannot contain more information about the true distribution than the data the generator was trained on

LLM-Generated Data

Four use cases

LLM-generated data is currently used for at least four distinct things:

1. **Instruction tuning data** (Alpaca, Self-Instruct) — generate (instruction, response) pairs to fine-tune a smaller model
2. **Distillation** — use a strong LLM as a teacher to label unlabeled inputs, train a smaller student
3. **Pretraining data** (phi series, "Textbooks Are All You Need") — generate synthetic textbook-quality text from a strong LLM, pretrain from scratch
4. **Augmentation for classical NLP** — paraphrase, perturb, or expand a small labeled dataset using an LLM as the rewriter

These look superficially similar but have **very different reliability stories**

Self-Instruct

The procedure:

1. Start with a small seed set of human-written instructions (~175)
2. Prompt a strong LLM (GPT-3.5/4) to generate new instructions in the same style
3. For each new instruction, prompt the LLM to generate an example input and output
4. Filter for quality and deduplicate
5. Result: **52K synthetic instruction-following examples**

Stanford Alpaca applied this to fine-tune LLaMA-7B → near-ChatGPT behavior at a fraction of the cost

Kicked off the wave of GPT-as-teacher datasets

Why Self-Instruct works (when it works)

- The teacher model has already done the hard work of learning instruction-following
- The student doesn't need to *discover* that knowledge — it just needs to **imitate** the teacher's outputs
- This is more like **knowledge distillation**, just packaged as synthetic data

Students inherit the teacher's:

- Capabilities (good)
- Style (mostly good)
- Biases (bad)
- Refusal patterns (subtle)
- **Errors and hallucinations** (very bad)

The False Promise

Gudibande et al. (2023): "The False Promise of Imitating Proprietary LLMs"

Trained students on outputs from strong teachers, evaluated carefully:

- **Style transfers easily** — the student sounds like the teacher
- **Factual correctness does not** — the student inherits the teacher's surface form but not its underlying knowledge

Imitation models look great on the benchmarks the teacher was good at; they regress to their pretraining quality on truly novel tasks

Imitation can close the **style gap** without closing the **capability gap**

Synthetic pre-training

Gunasekar et al. (2023): "Textbooks Are All You Need" → phi-1, then phi-1.5, phi-2, phi-3

Claim: a 1B-parameter model trained on filtered + LLM-generated "textbook-quality" data can match much larger models on coding and reasoning benchmarks

The most aggressive synthetic-data story to date — synthetic data as a substitute for **scale**

Caveat: the synthetic pretraining data was explicitly generated to cover the skills those benchmarks test. Phi looks great on HumanEval/MBPP because the training distribution was *designed to match the evaluation distribution* — on tasks outside that coverage, it performs like the 1B model it is.

Model collapse (Shumailov et al. (2024), *Nature*)

The claim: AI models **collapse** when trained on **recursively generated data**

The setup:

1. Train a model on real data
2. Use it to generate synthetic data
3. Train the next generation on that synthetic data
4. Repeat

The result: each generation **loses tails of the distribution**. Rare events disappear. Variance shrinks. After a few iterations, the model produces a degenerate, mode-collapsed distribution.

Why model collapse happens

The mechanism is purely **statistical**:

- Finite samples from a distribution **under-represent the tails**
- Models trained on those samples reproduce the under-representation
- The under-representation **compounds** across generations

Callback to **Lecture 8** (semi-supervised learning): this is **confirmation bias** at scale

- In self-training, a model labels its own unlabeled data, then trains on those labels
- Errors reinforce themselves — the model becomes more confident in its mistakes
- Model collapse is the same failure mode, just across generations instead of within one training run

The only escapes are the same ones we saw in Lecture 8: **anchor on real data** or **verify with an external signal**

Anchoring against collapse

Connect back to the spectrum:

Where you are on the spectrum	Collapse risk
Perturb real examples (flips, noise)	Very low — real distribution is the anchor
Back-translation	Low — real targets anchor the supervision
Generative augmentation + real data	Moderate — depends on mixing ratio
Pure self-generated synthetic data	**High** — no anchor
Recursive self-training across generations	**Severe** — collapse compounds

Most production uses are in the middle, anchored by some real data + filtering

Accumulate, don't replace

Gerstgrasser et al. (2024): "Is Model Collapse Inevitable?"

- Shumailov's setup *replaced* real data with synthetic each generation
- **This is the load-bearing assumption**
- If instead you **accumulate** — keep the original real data and *add* each generation of synthetic on top — collapse disappears
- Proven: test error has a **finite upper bound independent of the number of iterations** (linear-model analysis)
- Replicated across LMs, diffusion, and VAEs

The recursive-collapse horror story is about **fully replacing** with synthetic. Real pipelines that retain the original corpus are not doomed in the same way.

But: strong model collapse

Dohmatob et al. (2024), "Strong Model Collapse"

Pushes back on the optimistic story:

- Even a **tiny fraction** of synthetic data — as little as **1 in 1000** — can degrade scaling laws in certain regimes
- The curse isn't about *replacement*; it's about *contamination*
- As the web fills with LLM output, even training on "real" web data is increasingly training on synthetic

The picture is **less settled** than either Shumailov or Gerstgrasser alone suggests

The three positions

Position	Paper	Claim
Pessimistic	Shumailov (2024)	Recursive synthetic training inevitably collapses
Optimistic	Gerstgrasser (2024)	Collapse is an artifact of *replacing*; accumulate and you're fine
Qualified	Dohmatob / Feng (2024)	Even small synthetic fractions hurt scaling laws — but **verification** mitigates

For the course's low-resource setting, the practical recipe is clear:

> Keep real data, use synthetic as augmentation with filtering, anchor on **verification** when you can.

Verification as the escape hatch

Feng et al. (2024), "Beyond Model Collapse: Scaling Up with Synthesized Data Requires Verification"

Formalizes the pattern: **verification on synthesized data prevents collapse**

Working recipes in 2024-2025:

- **Llama 3**: synthetic data, but always anchored by real data and rigorous quality filtering
- **Reasoning models** (o1, DeepSeek R1): generate synthetic chain-of-thought, **filter by correctness** via verifiers
- **Code generation**: synthetic code filtered by unit-test pass rates
- **Math**: synthetic reasoning filtered by symbolic answer checkers

The pattern: **synthetic data works when there's an external verification signal**

Without verification, you're amplifying the generator's distribution

CoT-Self-Instruct (2025)

The method:

1. Instead of prompting the teacher to generate instructions directly...
2. Have the teacher **reason via chain-of-thought** about a seed task first
3. Then generate a new example of similar quality and complexity
4. **Self-filter via answer consistency**: keep only examples where the generated answer matches the LLM's majority-vote solution

Results:

- Beats vanilla Self-Instruct and **human-generated data** on AlpacaEval 2.0 / Arena-Hard
- Beats existing datasets on MATH500, AMC23, AIME24, GPQA-Diamond

The self-filtering step is the verifier — a cheap, model-internal one

Synthesis

The throughline

Every augmentation is an **inductive bias** — a claim about which features matter

The spectrum runs from:

- **Perturb real data** → anchored, low risk, modest gain
- **Generate from scratch** → unanchored, high risk, sometimes dramatic gain

The reliable pattern across the whole spectrum:

> The **side you care about** should be real; the side you can tolerate noise on is the one you generate.

The throughline, instantiated

Method	Real	Generated
SimCLR	Image	Augmented views
SpecAugment	Label (transcript)	Masked spectrogram
Back-translation	Target sentence	Source sentence
Distillation	Input	Teacher label
Verifier-checked CoT	Problem + answer	Reasoning trace

When **both** sides are synthetic → risk of collapse

Where this sits in the course

- **Lecture 3** (Inductive bias) — augmentations are explicit inductive bias; learned augmentations are meta-learned bias
- **Lecture 7** (Self-supervised) — the augmentation pipeline *is* the learning signal in contrastive methods
- **Lecture 8** (Semi-supervised) — consistency regularization = "predict the same thing under augmentation"
- **Lecture 11** (Domain adaptation) — sim-to-real and domain randomization are augmentation-as-domain-shift
- **Lecture 13** (Meta-learning) — AutoAugment learns the augmentation policy

Augmentation is the lens that ties most of the course together

For Tuesday's discussion

When reading augmentation/synthetic-data papers, ask:

1. What **invariance** is this augmentation hypothesizing? Is it correct for the task?
2. Where does the **supervision signal** actually come from — real labels, model labels, or self-labels?
3. Is there an **external verification** signal, or just a generator?
4. If the augmentation were removed, would the result still be impressive?