COSI-230B: Natural Language Annotation for Machine Learning

Lecture 18: Instruction Annotation & Instruction-Tuning Datasets

Jin Zhao

Brandeis University

April 13, 2026

Context

From Classification to Instruction Tuning

Setting the stage before we dive in

Same Models, Different Framing

What model are we using?

The same kind of neural network: a **Transformer**.

Traditional classification:

- BERT, RoBERTa
- One model per task
- `[CLS]` embedding → linear layer → softmax

Instruction tuning:

- GPT, LLaMA, T5
- One model for many tasks
- Generate output tokens autoregressively

Nothing magical changed in the architecture—the difference is in **what we treat as input** and **what we treat as supervision**.

What Actually Changes?

Traditional Classification

Format: `input` $\rightarrow$ `label`

Loss: $\text{CE}(\hat{y}_{\text{class}}, y_{\text{class}})$

Example:

Input: "This movie is great"

Output: `positive`

Predict one class from a fixed label set.

Instruction Tuning

Format: `instruction + input` $\rightarrow$ `text`

Loss: $\text{CE}(\hat{y}_{\text{tokens}}, y_{\text{tokens}})$

Example:

Input: "Classify sentiment: This movie is great"

Output: `positive` (generated token by token)

Predict a sequence of tokens, one at a time.

The only real difference

Instead of predicting a class, you predict a **sequence of tokens**. The "label" is now just text.

How It Works: Two Architectures

Decoder-Only (GPT / LLaMA)

Concatenate everything into one sequence:

`[Instruction + Input + Output]`

Training objective: predict the next token at every position.

Example:

"Classify sentiment: This movie is great $\rightarrow$ positive"

The model learns:
after "$\rightarrow$" $\Rightarrow$ predict "p"
after "p" $\Rightarrow$ predict "o"
. . . and so on.

Encoder-Decoder (T5 / FLAN)

Split into two parts:

Encoder input:
"Classify sentiment: This movie is great"

Decoder output:
"positive"

Still cross-entropy over output tokens—same loss, different information flow.

In both cases, supervision is **token-level cross-entropy** on the output sequence.

Why Does This Enable "Instruction Following"?

The model sees different instructions mapped to different behaviors:

Instruction + Input	**Target Output**
"Translate to French: Hello"	"Bonjour"
"Summarize: The committee met to..."	"The committee decided..."
"Classify sentiment: This movie is great"	"positive"

The model learns: **the instruction text determines what function to perform**.

Everything is unified as

`text` $\rightarrow$ `text`

There is no separate classifier head, no task-specific architecture—the "label" is just text: "positive", "yes", "Paris", "The main idea is..."

The Historical Shift: Why Instruction Tuning Now?

Before ∼2019 (Task-Specific Era)

- Models were small and task-specific
- One model for sentiment, one for QA, one for translation
- Could not generalize across tasks

After GPT-3 / Large LLMs

- Billions of parameters, massive pretraining
- Models *already know* many tasks implicitly
- But they don't know *which task you want right now*

The paradigm shift:

Old:	Train a model	→ for *one* task
New:	Train a model once	→ use instructions for *many* tasks

Instruction datasets became essential because models became **general-purpose** and we needed a way to **steer** them.

Why Not Just Prompting?

Without instruction tuning, prompting alone is brittle:

- Same task, different wording $\rightarrow$ wildly different results
- Requires careful prompt engineering for every use case
- No robustness to instruction variation

Instruction tuning fixes this by training on:

- Many tasks $\times$ many phrasings
- The model becomes robust to instruction variation

Why this only works with large models

Small models: not enough latent knowledge—instructions don't help much.
Large models: capabilities are already there—instruction tuning *activates and organizes* them.

The ability is already inside the model. Instruction tuning just **unlocks** it.

Why This Is Powerful—and Dangerous

Powerful

- One model handles many tasks
- No need to redesign the architecture
- "Labels" are natural language—infinitely flexible

Dangerous

The model may learn:

`instruction wording` $\rightarrow$ `output`

instead of:

`input meaning` $\rightarrow$ `output`

This is the **template leakage** problem.

Instruction datasets also connect to broader concerns:

- **Alignment:** Teaching models to behave according to human intent
- **Safety:** Preventing harmful or unintended behaviors
- **RLHF:** Reinforcement learning from human feedback builds on instruction tuning

The Deep Insight

Pretraining vs. Instruction Tuning

Pretraining:	Teaches the model *knowledge* and *language patterns*
Instruction tuning:	Teaches the model *how to behave when asked*

Instruction tuning is **not** about changing the model.
It is about changing **what the model treats as input vs. supervision**.

One clean takeaway

Bigger models made instruction tuning *possible*, but instruction datasets exist because we need a way to **steer** those models.

This is exactly what we will study today: how instruction datasets are designed, what makes them work, and what can go wrong.

Today's Agenda

Part I: Foundations

- Instruction annotation as task contract
- Anatomy of an instruction example
- Instruction schema design best practices

Part II: Paper Deep Dives

- Natural Instructions (Mishra et al.)
- Super-NaturalInstructions (Wang et al.)
- T0 and zero-shot generalization (Sanh et al.)
- Flan Collection (Longpre et al.)
- Self-Instruct & Alpaca
- LIMA: Less Is More

Part III: Artifacts & Quality Assurance

- Template leakage: theory + concrete example
- What happens when templates shift
- Instruction QA workflow
- Mitigations

Goal: Understand instruction data as task specification, not just prompt engineering.

Part I

Foundations of Instruction Annotation

Instruction Annotation = Specifying a Task Contract

The Conceptual Inversion

Classical annotation attaches labels *to data*.
Instruction annotation attaches *a task definition to many data*.

The hard part: Not writing one instruction—but designing **many variations**:

"Classify sentiment"
"What is the sentiment?"
"Is this positive or negative?"

"Instruction tuning does not magically confer generalization. It shifts the generalization problem: from unseen inputs to unseen tasks and formulations."

Anatomy of an Instruction Example

Example: Sentiment Classification

Task definition: Given a movie review, classify the sentiment as positive or negative.

Input: "This film was a masterpiece of storytelling and visual artistry."

Constraints: Output only "positive" or "negative". Do not explain.

Output: positive

Instruction schema (standardized fields):

- **Task definition:** What the model should do
- **Input:** The specific instance
- **Constraints:** Output format, length, style requirements
- **Examples** (optional): Few-shot demonstrations
- **Output:** The expected response

Mishra et al. (2022). Natural Instructions. ACL. `https://aclanthology.org/2022.acl-long.244/`

Instruction Schema Design: Best Practices

Component	Natural Instr.	Super-Natural	Flan	Self-Instruct
Task definition	Verbose, multi-sentence	Standardized fields	Per-template prompt	Seed-generated
Pos/neg examples	Yes, both included	Yes, both included	No (implicit)	No
Constraints	Explicit field	Explicit field	Embedded in prompt	Implicit
Output format	Free text	Category-specific	Category-specific	Free text
Template count	1 per task	1 per task	$\sim$10 per task	1 per task

Design principles:

1. **Separate intent from format:** The task definition should describe *what*, not *how to phrase it*
2. **Explicit constraints reduce ambiguity:** "Output only X or Y" prevents verbose answers
3. **Negative examples prevent over-generalization:** Show what *not* to do

Part II

Paper Deep Dives

Natural Instructions: Unified Schema for Many Tasks

Mishra et al. (2022): Benchmark generalization to *new tasks* given natural language instructions.

Key design decisions:

- Standardized instruction format
- Positive and negative examples
- Explicitly tests *cross-task* generalization
- Tasks from existing NLP datasets

What it revealed:

- Models can follow novel instructions
- Performance depends heavily on instruction wording
- Template/format choices leak into predictions
- Few-shot examples help but introduce their own biases

Mishra et al. (2022). Natural Instructions: Benchmarking Generalization to New Tasks from Natural Language Instructions. ACL. https://deepai.org/publication/natural-instructions-benchmarking-generalization-to-new-tasks-from-natural-language-instructions

Natural Instructions: What Does an Entry Look Like?

Task: Question Generation from a Passage

Definition: Given a passage about a topic, generate a question whose answer can be found in the passage. The question should require understanding, not just copying a sentence.

Positive example:
Input: "The Eiffel Tower was built in 1889 for the World's Fair in Paris."
Output: "Why was the Eiffel Tower built?"
Explanation: The question requires understanding the purpose, and the answer is in the passage.

Negative example:
Input: "The Eiffel Tower was built in 1889 for the World's Fair in Paris."
Output: "What is the Eiffel Tower made of?"
Explanation: The answer is **not** in the passage—this is a bad question.

Key feature: **verbose definitions** + **positive and negative examples with explanations**.

Natural Instructions: Key Findings

Dataset: 61 tasks drawn from existing NLP benchmarks (QA, classification, generation, etc.)

Quantitative findings:

- GPT-3: $\sim$55% RougeL with instructions (vs. $\sim$22% without)
- $+2$ positive examples $\Rightarrow$ $+12$ points on average
- $+$negative examples $\Rightarrow$ further $+4$ points
- Variance across tasks is very high ($\sigma > 20$ pts)

Annotation takeaways:

- Quality matters more than length
- Negative examples: under-used but informative
- Task decomposition consistently helps
- Human agreement on clarity is moderate ($\kappa \approx 0.6$)

Lesson for annotators

Writing instructions is itself an annotation task that requires calibration, guidelines, and inter-annotator agreement measurement.

Super-NaturalInstructions: Scaling Instruction Generalization

Wang et al. (2022): Scale to 1600+ NLP tasks with declarative instructions.

- Tests **generalization targets**: unseen tasks vs. unseen templates vs. unseen languages
- Shows that **instruction coverage** matters more than model size (up to a point)
- Formalizes "instruction generalization" as a benchmark

Generalization Target Taxonomy

Unseen tasks	New task types not in training
Unseen templates	Same task, different instruction wording
Unseen languages	Same task, different language

Annotation implication: How you split train/test determines what "generalization" means.

Wang et al. (2022). Super-NaturalInstructions. EMNLP. https://aclanthology.org/2022.emnlp-main.340/

Super-NaturalInstructions: What Does an Entry Look Like?

The same structured schema, but now for **1,616 diverse tasks** across **55 languages**:

Task 672: Word Analogy

Definition: Given a word analogy "A is to B as C is to ?", provide the answer.
Input: "hot is to cold as big is to ?"
Output: "small"

Task 1391: Sentence-to-Code (Hindi)

Definition: Convert the Hindi sentence into a Python expression.
Input: "do aur teen ka yog" [*Hindi*: sum of 2 and 3]
Output: 2 + 3

Compare to Natural Instructions

Same schema format, but now the **breadth** is the point: analogies, code generation, cross-lingual tasks, creative writing, math... all in one dataset.

Super-NaturalInstructions: Key Findings

Scale: 1,616 tasks, 76 task types, 55 languages, community-contributed via GitHub

Quantitative findings:

- Tk-Instruct (11B) trained on 757 tasks outperforms InstructGPT (175B) on held-out tasks
- Performance scales log-linearly with number of training tasks up to $\sim$800 tasks
- Cross-lingual transfer: 83% of English-only performance when tested on non-English tasks
- Diminishing returns beyond $\sim$64 examples per task

Community annotation process:

- Crowd-sourced via structured schema on GitHub
- Each task requires: definition, 2+ positive examples, 2+ negative examples, constraints
- Automated validation checks on submission
- Reveals tension: broad coverage vs. quality control at scale

Super-NaturalInstructions: Key Findings (cont.)

Key insight

Task diversity matters more than per-task data volume. 100 diverse tasks beat 10 tasks with $10\times$ data each.

T0 and the Zero-Shot Generalization Story

Sanh et al. (2022): Multitask prompted training enables zero-shot task generalization.

Design:

- Fine-tune T5-11B on a diverse mixture of prompted datasets (36 datasets, 62 tasks)
- Each task has multiple human-written prompt templates via PromptSource
- Evaluate zero-shot on 11 held-out datasets

Key results:

- T0 (11B) matches or exceeds GPT-3 (175B) on 9 of 11 held-out tasks
- Prompt template diversity is critical: more templates $\Rightarrow$ better generalization
- Median performance drops $\sim$15 points when using worst vs. best template per task

PromptSource annotation:

- Collaborative tool for writing and sharing prompt templates
- 2,073 prompts across 177 datasets
- Templates written by NLP researchers (not crowdworkers)
- Each template maps dataset fields to natural language

T0 / PromptSource: What Does an Entry Look Like?

The key idea: the **same data**, but with **multiple prompt templates**.

Underlying data (NLI — from MNLI):
Premise: "A man is playing guitar on stage."
Hypothesis: "A man is performing music." Label: `entailment`

Template 1

"A man is playing guitar on stage." Based on this, is it true that "A man is performing music"?

Output: "Yes"

Template 2

Does "A man is playing guitar on stage" entail "A man is performing music"?

Output: "Yes, it entails it"

Template 3

Suppose "A man is playing guitar on stage." Can we conclude "A man is performing music"?

Output: "Yes"

Same task, same data, **3 different ways to ask**. The model must learn the *task*, not memorize one template.

T0 and the Zero-Shot Generalization Story (cont.)

Template sensitivity

A single task can range from 30% to 80% accuracy depending on which template is used—template choice is a first-class annotation decision.

Sanh et al. (2022). Multitask Prompted Training Enables Zero-Shot Task Generalization. ICLR.
`https://arxiv.org/abs/2110.08207`

Flan: Design Decisions That Matter

Longpre et al. (2023): The Flan Collection—designing data and methods for effective instruction tuning.

Engineering decisions that affect outcomes:

1. **Task balancing:** Upsampling rare tasks, downsampling dominant ones
2. **Template diversity:** Multiple prompt templates per task
3. **Enrichment:** Adding chain-of-thought variants, few-shot examples
4. **Mixing ratios:** Proportion of zero-shot vs. few-shot vs. CoT

Key Evidence

Instruction tuning outcomes depend heavily on *mixing and enrichment decisions*—this is "annotation mixture engineering," not just training.

Longpre et al. (2023). The Flan Collection. arXiv:2301.13688. https://arxiv.org/abs/2301.13688

Flan: What Does an Entry Look Like?

Flan's key innovation: the **same task** appears in three different modes.

Task: Natural language inference

Zero-Shot

Input: "Premise: She went to the store. Hypothesis: She bought groceries. Does the premise entail the hypothesis?"

Output: "It is not possible to tell"

Few-Shot

Input: "Premise: He ran fast. Hyp: He was moving. Answer: Yes.
Premise: She slept. Hyp: She was awake. Answer: No.
Premise: She went to the store. Hyp: She bought groceries. Answer:"

Output: "It is not possible to tell"

Chain-of-Thought

Input: "She went to the store. Does this mean she bought groceries? Think step by step."

Output: "Going to the store does not necessarily mean buying groceries. She could be browsing. So: it is not possible to tell."

The model trains on **all three modes**—50% zero-shot, 25% few-shot, 25% CoT.

Flan: Ablation Results and Numbers

Flan 2022 merges Flan 2021, P3 (T0's data), Super-NaturalInstructions, and new dialog/CoT tasks.

Key ablation results:

- Mixing 1,836 tasks: +9.4% over Flan 2021 on held-out MMLU
- Adding CoT data: +2.1% on reasoning benchmarks (BBH)
- Template diversity (10 per task vs. 1): +4.2% on average
- Few-shot enrichment: +1.8% beyond zero-shot only
- Task balancing via examples-proportional mixing: +2.5% on rare task types

Mixture recipe:

- 50% zero-shot prompted
- 25% few-shot prompted
- 25% chain-of-thought
- Examples-proportional mixing with cap at 30K per task
- Input inversion: ∼10% of examples have input/output swapped for robustness

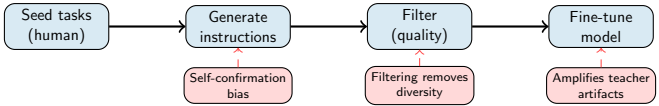
Flan: Ablation Results and Numbers (cont.)

Takeaway

Instruction tuning is as much about *data engineering* (mixing, balancing, enrichment) as it is about the model or the training algorithm.

Self-Instruct: Synthesize–Filter–Fine-Tune

Wang et al. (2023): Aligning Language Models with Self-Generated Instructions.



Where artifacts creep in:

- Prompting seeds shape the task distribution (seed bias)
- Filtering for quality/validity removes diversity
- Generated outputs inherit teacher model's style and errors

Wang et al. (2023). Self-Instruct. ACL. `https://aclanthology.org/2023.acl-long.754/`

Self-Instruct: What Does the Pipeline Produce?

Human-Written Seed Task

Instruction: "Determine whether the given sentence is grammatically correct."
Input: "She don't like pizza."
Output: "No, it is not grammatically correct."

$\Rightarrow$ GPT-3 generates

Machine-Generated Task

Instruction: "Identify and correct the grammatical error in the following sentence."
Input: "Him and me went to the movies."
Output: "He and I went to the movies."

Notice: the generated task is *related* but *different*—it shifted from judgment to correction. This is how 175 seeds become 52K tasks. But also how **style and phrasing patterns** propagate.

Self-Instruct: Detailed Findings

Pipeline numbers:

- 175 human-written seed tasks $\rightarrow$ 82K generated instructions (after filtering)
- ROUGE-L filtering threshold: discard if > 0.7 overlap with existing instructions
- 52K instructions survive quality filtering ($\sim$63% pass rate)
- Human evaluation: 54% of generated instructions are valid and novel

Quality analysis:

- Valid instruction + correct output: 54%
- Valid instruction + incorrect output: 12%
- Invalid / nonsensical instruction: 34%
- Distribution heavily skewed toward classification and generation tasks

Impact on downstream model:

- GPT-3 + Self-Instruct tuning nearly matches InstructGPT on Super-NaturalInstructions
- Biggest gains on creative generation tasks
- Smallest gains on tasks requiring factual accuracy
- Seed task diversity directly predicts output diversity

Wang et al. (2023). Self-Instruct: Aligning Language Models with Self-Generated Instructions. ACL

Alpaca: "Semi-Synthetic Instruction Tuning"

Taori et al. (2023): Use GPT-3.5 to generate 52K instruction–output pairs from 175 seed tasks, then fine-tune LLaMA.

Pipeline:

1. Start with 175 human-written seed tasks
2. Prompt GPT-3.5 to generate new tasks + outputs
3. Light filtering for format/length
4. Fine-tune LLaMA-7B on result

Where annotation artifacts enter:

- Seed task distribution shapes output distribution
- GPT-3.5's style becomes the "ground truth"
- No human verification of generated outputs
- Template artifacts from the generation prompt

Bridge to Module 5 (Synthetic Annotation): Alpaca shows synthetic data is partly annotation, partly generation.

Taori et al. (2023). Alpaca: A Strong, Replicable Instruction-Following Model. Stanford CRFM.

Alpaca: What Does an Entry Look Like?

Alpaca uses a simple three-field format—all generated by GPT-3.5:

Example 1: With input field

Instruction: "Classify the following sentence as either a question or a statement."
Input: "Have you ever been to Paris?"
Output: "This sentence is a question."

Example 2: No input field (open-ended)

Instruction: "Give three tips for staying healthy."
Input: (empty)
Output: "1. Eat a balanced diet... 2. Exercise regularly... 3. Get enough sleep..."

Notice the characteristic **GPT-3.5 style**: polished, verbose, list-heavy. This style becomes the "ground truth" that LLaMA learns to imitate.

LIMA: Less Is More for Alignment

Zhou et al. (2023): Fine-tune LLaMA-65B on only **1,000 carefully curated examples**.

Key results:

- 1,000 examples $\approx$ 52K Alpaca examples in human preference
- LIMA preferred over Alpaca 73% of the time
- Ties or beats GPT-4 (text-davinci-003) on 43% of prompts

Data composition:

- 750 from Stack Exchange / wikiHow / Reddit (top answers)
- 250 manually written by the authors
- Zero synthetic data

The "Superficial Alignment Hypothesis"

A model's knowledge and capabilities are learned almost entirely during pretraining. Alignment tuning teaches the model the *format* and *style* of interaction, not new knowledge.

LIMA: What Does an Entry Look Like?

Alpaca (52K synthetic, GPT-3.5)

Q: "Explain how a car engine works."

A: "A car engine works by converting fuel into mechanical energy. The engine uses a four-stroke cycle: intake, compression, combustion, and exhaust. . . "

Generic, textbook-like, written by GPT-3.5.

LIMA (1K curated, human-selected)

Q: "Why does my car shake when I brake at high speed?"

A: "Most likely warped brake rotors. When you brake hard at highway speed, friction heats the rotors unevenly. You'll feel a pulsing in the pedal. A mechanic can measure rotor runout—if it's over 0.002", they need resurfacing or replacement."

Specific, practical, expert-level—from a top Stack Exchange answer.

The LIMA philosophy

1,000 responses like the right column teach better behavior than 52K responses like the left.

LIMA: Less Is More for Alignment (cont.)

Annotation implication:

- Quality $\gg$ quantity for instruction data
- Careful curation of 1K examples can rival 52K noisy ones
- Selection criteria matter more than generation volume

Zhou et al. (2023). LIMA: Less Is More for Alignment. NeurIPS. https://arxiv.org/abs/2305.11206

Part III

Artifacts and Quality Assurance

Core Artifact: Template Leakage

Definition

Template leakage: When instruction tokens (template text) correlate with the label, a model can learn "what template implies" rather than "what the input means."

Example:

- Template A (for positive): "Say POSITIVE:" + review text
- Template B (for negative): "Say NEGATIVE:" + review text
- A classifier can achieve high accuracy by reading only the template!

Why decoder-only LMs are especially sensitive:

- Instruction text and output share the same token sequence
- Attention can freely attend to template tokens
- Spurious correlations in training are amplified at scale

Mitigation: Randomize templates, test with paraphrased instructions, audit template–label correlations.

Template Leakage: A Concrete Walkthrough

Scenario: NLI dataset with 3 labels. An annotator writes one template per label:

Templates used during instruction tuning

Entailment: "Given the premise, does the hypothesis follow? Answer yes."
Contradiction: "Given the premise, does the hypothesis conflict? Answer no."
Neutral: "Given the premise, is the hypothesis related? Answer maybe."

Token–label correlation analysis:

Token	PMI with label	TF-IDF rank	Problem
"follow"	1.0 (entailment)	#1	Perfectly predictive
"conflict"	1.0 (contradiction)	#1	Perfectly predictive
"related"	1.0 (neutral)	#1	Perfectly predictive
"yes"	1.0 (entailment)	#2	Label embedded in template
"no"	1.0 (contradiction)	#2	Label embedded in template

Template Leakage: A Concrete Walkthrough (cont.)

Result

A model trained on this data achieves $>95\%$ accuracy by attending to template tokens alone. On a *neutral* template ("Classify the relationship:") accuracy drops to $\sim 40\%$.

Why Decoder-Only LMs Are Sensitive to Formatting

Encoder-only (e.g., BERT):

- Input and output are separated
- Classification head ignores instruction format
- Less susceptible to template leakage

Decoder-only (e.g., GPT, LLaMA):

- Instruction + output in one sequence
- Autoregressive attention sees all prior tokens
- Template tokens directly influence generation

Practical Consequence

Format adherence vs. task quality becomes a real tradeoff. Instruction schema choices (input/output format, constraints, examples) directly shape modeling outcomes.

What Happens When Templates Shift?

Experiment 1: Leaky Templates

Setup: Train with label-coded templates ("Say POSITIVE:" / "Say NEGATIVE:")

Test condition	Accuracy
Same leaky templates	>95%
Neutral template	∼50%

The model learned the **template**, not the task.

Experiment 2: Single Template

Setup: Train with only one phrasing ("Classify the sentiment of this review:")

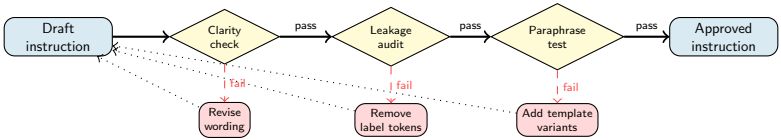
Test condition	Accuracy
Same template	∼95%
Paraphrased templates	∼60%

Even without leakage, **one template** creates fragility.

Takeaway

Template diversity is not optional—it is a **requirement** for robust instruction-tuned models.

Instruction QA Workflow



Stage details:

1. **Clarity check:** Can a non-expert understand the task from the instruction alone? (Human eval, 2 reviewers)
2. **Leakage audit:** Compute PMI between template tokens and labels. Flag if any token has PMI > 0.5 with a single label.
3. **Paraphrase test:** Rewrite the instruction 3 ways. If performance drops $>10\%$ on paraphrased versions, the instruction is over-fitted.
4. **Approval:** Instruction enters the final dataset with all template variants.

Mitigations for Template Leakage

1. **Template randomization:** Use multiple templates per task; ensure no template–label correlation
2. **Paraphrase tests:** Evaluate with rephrased instructions to test robustness
3. **Mixture audits:** Check task/template proportions; avoid systematic imbalances
4. **Cross-template evaluation:** Train on template A, test on template B
5. **Instruction QA workflow:**
   - Human reviewers check instructions for clarity and leakage
   - Edge-case checklists: adversarial inputs, boundary cases
   - Automated checks: token overlap between template and label

Longpre et al. (2023) show that Flan's template mixing improves robustness. `https://arxiv.org/abs/2301.13688`