



Semester Project 2

Master of Science in Engineering

School of Engineering - Centre for Artificial Intelligence

Do NOT Classify and Count

Hybrid Attribute Control Success Evaluation

A Technical Report

Author

Felix Matthias Saaro

Supervisor

Prof. Dr. Mark Cieliebak

Secondary Supervisors

Dr. Jan Milan Deriu
Pius von Däniken

July 31th, 2025

Abstract

Attribute control in controllable generation is typically evaluated using pretrained classifiers. It has been shown that the common Classify & Count (CC) method leads to biased and inconsistent results. Estimates are found to vary significantly across different classifiers. In this project, Attribute Control Success (ACS) estimation is framed as a quantification task. A hybrid Bayesian method, called Bayesian Classify & Count (BCC), is applied, in which classifier predictions are combined with a small number of human labels used for calibration.

To evaluate the method, a dual modality benchmark containing both text and image samples was collected. It consists of 600 human annotated samples and 60'000 metric annotated samples. Through experiments, it is shown that the BCC method produces robust estimates across both text and text-to-image generation tasks. It is also shown that the BCC method enables consistent pairwise comparisons of model performance across different classifiers, yielding stable rankings among generators.

Furthermore, the information gain from metric annotations is quantified, highlighting the added value of human annotations over metric-based ones. Thus, a more reliable and principled alternative to existing evaluation practices is provided.

As part of this project, a toolkit was developed that can be used—either in parts or as a whole—to estimate ACS on various generative models. Support is provided for both text and text-to-image generation in combination with binary attribute control.

Preface

This technical report accompanies the paper titled “*Do NOT Classify and Count: Hybrid Attribute Control Success Evaluation*” (to be published), written as part of this project. The report gives a deeper look into the methods, dataset, tools, and results. It explains the custom toolkit developed to run experiments and gather results. It also includes background information in simple terms to help readers understand this complex topic. In addition to the technical report and the paper a code repository ¹ is provided with the developed toolkit as well as extra code snippets to (i) reproduce the results and (ii) apply to your own controllable generation model.

Felix Saaro
Winterthur, July 31st 2025

¹<https://github.com/MrF3lix/generative-model-evaluation-toolkit>

Glossary

ACS Attribute Control Success. i, 1, 4, 9

BCC Bayesian Classify & Count. i, 2, 3, 14, 20, 22, 24, 32, 34–41

CC Classify & Count. i, 1–3, 11, 12, 14, 20, 22, 24, 32, 34–37, 39–41

CG controlled generation. 1, 2, 9

CPCC Calibrated Probabilistic Classify & Count. 20

CTG controlled text generation. 2, 3

EAS effective additional sample. 25

FP false positive. 12

FPR False Positive Rate. 1–3, 12, 13, 28, 31

HF HuggingFace. 15, 18, 19, 26

JS Jensen-Shannon. 24, 32, 34, 41

LLM large language models. 2, 20, 26

MCMC Markov Chain Monte Carlo. 13, 22, 25

PPLM Plug and Play Language Model. 3

TP true positive. 12

TPR True Positive Rate. 1–3, 12, 13, 28, 30, 31

VQA visual question answering. 3

Contents

1	Introduction	1
1.1	Contributions	2
1.2	Related Work	2
2	Methods	4
2.1	Annotation	4
2.1.1	Annotation Tool	4
2.1.2	Annotator Guidelines	5
2.1.2.1	Sentiment Stories	5
2.1.2.2	Animal Images	7
2.1.3	Cohen’s Kappa	8
2.2	Evaluate Attribute Control Success	9
2.2.1	Beta Distributions	10
2.2.1.1	Example	11
2.2.2	Classify & Count	11
2.2.3	Bayesian Classify & Count	12
2.2.3.1	Confusion Matrix	12
2.2.3.2	Model Specification	13
2.3	Evaluation Toolkit	14
2.3.1	Data Structures	14
2.3.2	Generate	15
2.3.2.1	Configuration	15
2.3.2.2	Input	16
2.3.2.3	Output	16
2.3.3	Annotate	17

2.3.3.1	Configuration	17
2.3.3.2	Requirements	18
2.3.3.3	Input	18
2.3.3.4	Output	18
2.3.3.5	Current Limitations	18
2.3.3.6	Evaluate	18
2.3.3.7	Configuration	18
2.3.3.8	Input	19
2.3.3.9	Output	20
2.3.4	Quantify	20
2.3.4.1	Configuration	20
2.3.4.2	Input	21
2.3.4.3	Output	21
2.3.4.4	Current Limitations	22
2.4	Experimental Setup	22
2.4.0.1	Beta Moment Matching	22
2.4.0.2	Experiment Pairs	22
2.4.0.3	Experiment Overview	23
2.4.0.4	Analysis Goals	23
2.4.1	Consistency	24
2.4.2	Measurability	24
2.4.3	Sample Value	25
3	Data	26
3.1	Sentiment Stories	26
3.1.1	Generators	26
3.1.2	Classifiers	26

3.1.3	Human Annotations	27
3.1.4	Corpus Statistics	27
3.2	Animal Images	29
3.2.1	Generators	29
3.2.2	Classifiers	30
3.2.3	Human Annotations	30
3.2.4	Corpus Statistics	30
4	Results	32
4.1	Consistency	32
4.2	Measurability	35
4.3	Sample Value	38
5	Conclusion	39
5.1	Future Work	39

1 Introduction

Controlled generation (CG) systems have advanced rapidly in recent years, yet their evaluation remains challenging.

CG refers to the task of producing content that satisfies user-specified attributes. For text, these attributes may be semantic, structural, or lexical [1]. For images, the goal is typically to match a textual description, render a specific subject, or reflect a stylistic attribute [2]. The development of large pretrained models has led to major improvements in these tasks.

Modern text generation methods control attributes using prompting, guidance, or auxiliary reward models, rather than by changing the model architecture [3, 4, 5, 6]. At the same time, diffusion-based models have become the dominant approach for text-to-image generation [7, 8].

CG systems can be evaluated along various dimensions. In text, this includes fluency and coherence [9]; in images, visual fidelity is typically considered [10]. In this work, the focus is placed on Attribute Control Success (ACS), which asks whether generated outputs satisfy a given control attribute. For text generation, this might involve controlling sentiment. For image generation, it may require producing scenes with a specific number of objects. ACS is usually defined as the proportion of outputs that satisfy the intended condition.

Because full human evaluation of ACS is unfeasible, because it is time consuming and expensive, pre-trained classifiers are often used as a proxy (see Section 1.2). In such cases, the proportion of outputs classified as matching the desired attribute is reported. This method is known as Classify & Count (CC) in the quantification literature. There, the aim is to estimate the prevalence of a target property across a collection of items, rather than to classify individual instances. The overall ACS in a generated dataset is one such example. As shown by [11], the CC method often leads to biased results. These distortions depend on the True Positive Rate (TPR) and False Positive Rate (FPR) of the classifier used.

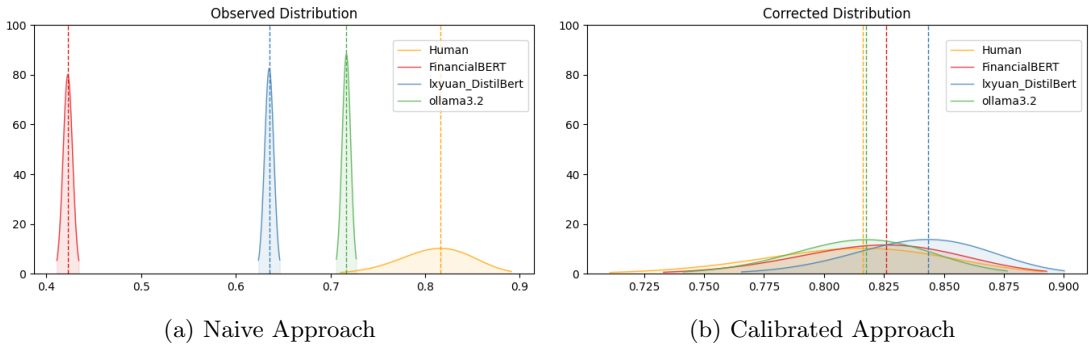


Figure 1: Motivating Example. Letting a generator write 10’000 stories controlled by a sentiment label, and then letting sentiment classifiers check if the sentiment was correctly expressed. The automated ratings are then compared to 100 human judgments.

To illustrate the issue, Figure 1a is considered. A language model was prompted to generate 10’000 short texts, each conditioned on a specific sentiment. Control success was evaluated in two ways: first, by collecting human annotations for a random sample of 100 generations; second, by applying three off-the-shelf sentiment classifiers to the full dataset (see Section 3 for details). For each method, the distribution

of control success rates was estimated, which is the frequency with which the generated texts matched the requested sentiment.

As shown in the Figure 1a, classifier based estimates exhibit low variance due to the large sample size. However, these estimates differ significantly from one another and from the human baseline. Each classifier produces a different success rate, revealing inconsistency. The human estimate, though noisier because of the small sample, presents a fundamentally different picture. This suggests that the CC method is inadequate in this context.

In this work, a hybrid quantification method is applied [12], called Bayesian Classify & Count (BCC). This method combines the 100 human annotations with classifier predictions over the entire dataset. As illustrated in Figure 1b, this approach yields control success estimates that are more consistent across classifiers and more closely aligned with the human distribution.

Although this hybrid approach requires a small number of human annotations, this supervision is considered essential. It helps bridge the domain gap between the classifier’s training data and the synthetic outputs generated by controlled systems. Alternative quantification methods, which adjust CC estimates [11, 13], rely on accurate knowledge of classifier performance. Specifically, TPR and FPR rates on the target data are required. In practice, such information is rarely available, since the distribution of generated text often differs substantially from the classifier’s original training domain. [14] emphasize the strong stability assumptions needed to extrapolate classifier behavior to new domains. These assumptions are frequently violated in practice, resulting in unreliable estimates.

1.1 Contributions

In this work, metric calibration for CG evaluation is formalized by adapting the BCC method proposed by [12]. To demonstrate the effects of calibration, a dual modality benchmark was created. It includes 600 human annotated samples and 60’000 metric annotated samples. It is shown that applying calibration leads to consistent evaluation results. Also included is the toolkit to apply the adapted methods to other CG evaluations.

1.2 Related Work

Whether the output is text or an image, the standard method for checking attribute control is the same: a pretrained classifier or detector is used, and positive cases are counted. In controlled text generation (CTG), this often involves sentiment, topic, or toxicity classifiers. More recently, zero-shot LLMs have also been used [1, 9]. For text-to-image generation, object detectors such as YOLO [15] or DETR [16], or VQA [17] models like *VQAScore* and *TIFA*, are typically applied [17, 18].

Benchmark datasets for CTG often rely on this approach. For example, [19] introduce *CoDI-Eval*, a dataset of diverse instruction prompts, and evaluate control success using popular classifiers from *HuggingFace*². Similarly, the *ConGenBench* dataset [20] is built to support classifier based evaluation.

²<https://huggingface.co>

In practice, CTG systems are also evaluated using classifiers. Gehman et al. [21] propose the *RealToxicityPrompts* benchmark and estimate toxicity using *PerspectiveAPI*³. Dathathri et al. [3] evaluate their Plug and Play Language Model (PPLM) using both human and classifier annotations. A notable gap is reported between sentiment control accuracy measured by humans and by classifiers, though this is not analyzed in detail.

Ke et al. [22] bypasses external classifiers with *CTRLEval*, which frames evaluation as a text-infilling task. The method assigns higher control scores to generations where attribute specific tokens have higher summed probabilities. This score is validated through correlation with human Likert ratings.

For text-to-image generation, Hartwig et al. [10] provides a detailed overview of evaluation metrics. Content based metrics are particularly relevant to this work, especially those checking whether generated images contain attributes described in the text prompt, such as *VQAScore* [17].

Binyamin et al. [23] evaluates object count control using both human annotations and the *YOLOv9* [24] object detector. A small difference is observed between human and detector-based counts. Hu et al. [18] proposes *TIFA*, a benchmark and evaluation framework using visual question answering (VQA). Both VQA and object detection are conceptually similar to classifier based evaluation in CTG.

The problem of estimating how often a target attribute appears in a dataset is known as *quantification*. The practice of applying a classifier to each item and reporting the class proportion is called Classify & Count (CC). Forman [11] studies CC in binary quantification and shows that it produces biased results, influenced by the classifier’s TPR and FPR. Corrected methods are proposed to address this. Bella et al. [13] introduces a probabilistic version of CC that uses classifier confidence scores. Sentiment quantification has also featured in SemEval shared tasks [25, 26].

In general, classifier based quantification depends on understanding how the classifier behaves on the target data specifically its TPR, FPR, or calibration curve [14]. Because the target data often differs from the classifier’s training data, domain shift is introduced. To address this, a hybrid approach is needed, where a small set of human labels anchors the quantification.

Von Dänkien [12] shows that accurate quantification can be achieved using just 100 in-domain human labels. This is done through a hybrid Bayesian Classify & Count (BCC) method, previously applied in machine translation evaluation [27, 28]. The current work relies on BCC as an alternative to naive CC.

³<https://www.perspectiveapi.com/>

2 Methods

The methods used in this project cover a wide range of tasks, from generating a benchmark datasets and managing annotation workflows to developing a toolkit for quantification and analyzing distribution consistency. This section provides an overview of the key processes and tools used. Special focus is placed on the annotation procedures, the creation of benchmarks that combine human and metric annotations, and the implementation of hybrid quantification methods designed to improve Attribute Control Success (ACS) estimation.

2.1 Annotation

Two datasets were required for this project. No existing dataset met the specific needs, which included the generation condition, human annotations, and metric annotations. This section describes the methods used to obtain the human annotations.

2.1.1 Annotation Tool

The tool Prodigy [29] was used to annotate both datasets. It was chosen because it supports different data types such as text and images.

A custom script, called a receipt, was created to allow annotation of two variables per generated item. This setup ensured that annotators only needed to annotate each item once.

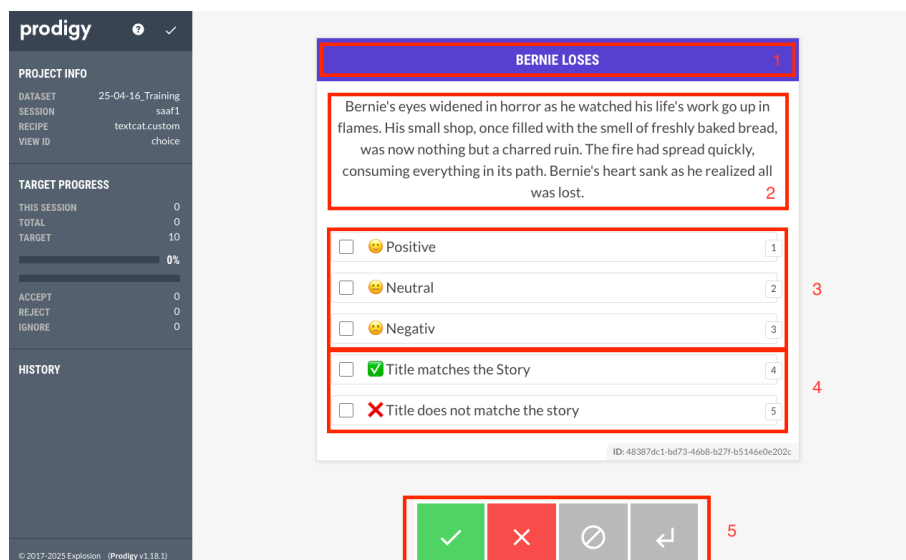


Figure 2: Overview of the annotation tool Prodigy. Showing a generated story including the title and the labels to set by the annotator. Two variables are annotated in this scenario, the sentiment (Positive, Neutral, Negative), and whether the title matches the story or not.

Figure 2 shows the annotation interface for the Sentiment Story dataset. The annotators see the title, also called the condition, and the generated story. They annotate the perceived sentiment (Positive, Neutral, Negative) and whether the title matches the story.

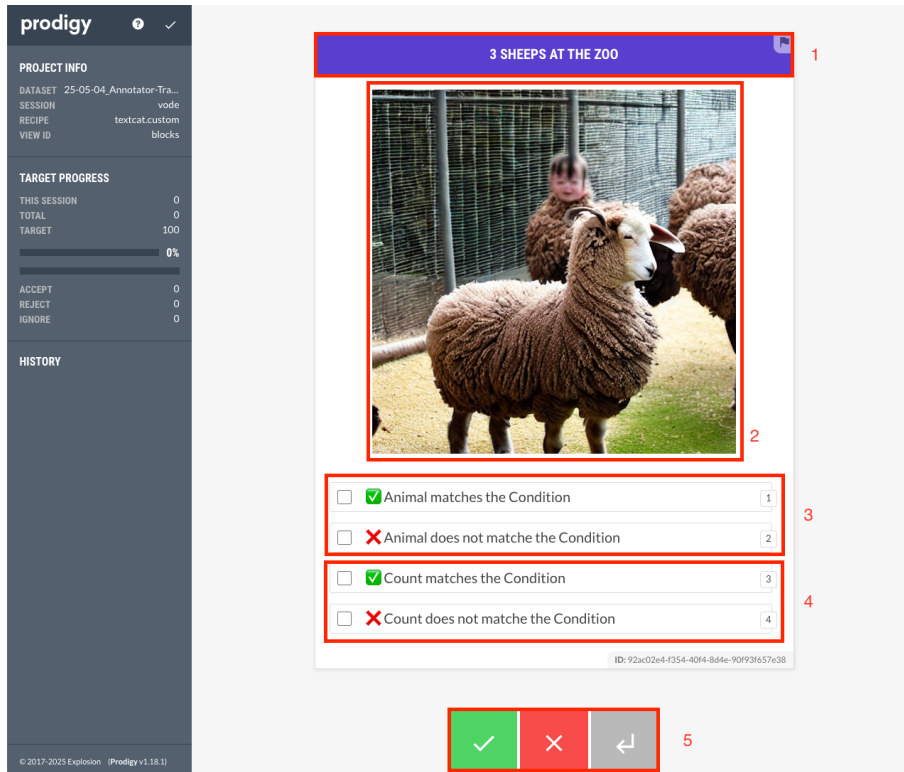


Figure 3: Overview of the annotation tool Prodigy. Showing a generated image including the condition and the labels to set by the annotator. Two variables are annotated in this scenario, whether the animal type matches the condition and whether the animal count matches the condition.

Figure 3 shows the annotation interface for the Animal Images dataset. The annotators see the prompt, also called the condition, and the generated image. They annotate whether the animal type matches the condition and whether the number of animals matches the condition.

2.1.2 Annotator Guidelines

For each annotation task guidelines were created to ensure consistent annotations among the annotators.

2.1.2.1 Sentiment Stories

Task In this task, you will annotate AI-generated short stories (< 5 sentences). Each story was generated from a *title* and a target *sentiment* (POSITIVE, NEUTRAL, NEGATIVE). Your job is to decide (i) whether the title matches the story and (ii) whether the target sentiment is reflected in the story.

Objectives

1. Annotate each short story with the **expressed sentiment**.
2. Annotate **whether the title matches** the short story (title appears in the purple box, story underneath).

Sentiment definition

Sentiment is the emotional tone or attitude expressed by—or toward—a character, event, or situation.

Label	Description
POSITIVE	Clear positive sentiment (praise, satisfaction, relief).
NEGATIVE	Clear negative sentiment (complaint, sorrow, anger).
NEUTRAL	Neither clearly positive nor negative; factual or ambiguous.

Table 1: Sentiment labels

Contextual considerations

- **Domain context** “The *battery* died quickly” → Negative; “The *killer* died quickly” → Neutral or Positive.
- **Sarcasm / irony** If sarcasm is obvious, label by the intended sentiment: “Great, another crash. Just what I needed!” → Negative.
- **Negation flips polarity** “The design is *not* bad” → Positive; “Not happy with the service” → Negative.
- **No affect.** If no emotional tone is expressed, choose NEUTRAL. “The phone was released in 2020.” → Neutral.

Examples

Title	Generated Story	Title Match?	Sentiment
The Accident	Sarah’s car skidded on wet pavement and crashed into a lamppost. She was shaken but unharmed.	Yes	Neutral
Keith wants to be a star	Keith’s obsession with fame left him empty and unfulfilled as city lights mocked his failure.	Yes	Negative
Albert the Elephant	Bard the Giraffe’s voice cracked; animals mocked him until he never sang again.	No	Negative

Table 2: Illustrative annotation examples.

Annotation tips

- Focus on *intent* rather than surface keywords.

- Read the entire story to grasp context before labeling.
- When in doubt, flag the item for discussion during adjudication.

2.1.2.2 Animal Images

Task

You will annotate AI-generated images created from prompts that specify three conditions:

1. **Animal count** (*how many* animals),
2. **Animal type** (*which* animal),
3. **Environment** (background setting).

Your goal is to decide whether the *animal count* and *animal type* in the prompt, match what appears in the generated image.

Objectives

1. Label whether the **animal type is correct**.
2. Label whether the **animal count is correct**.

Definitions

- **Animal type** matches if the depicted animal is unmistakably identifiable. Proportional errors (e.g. short neck, uneven legs) are ignored.
- **Animal count** is the number of clearly identifiable animals of that type. An instance *does not count* if (i) the animal is not clearly identifiable, or (ii) The majority of its body lies outside the image frame.

Ambiguity

If the image is ambiguous, click the *flag* icon (top-right).

Annotation interface.

See Figure 3.

1. Prompt box (shows the conditions, including the relevant conditions C1 and C2)
 - C1* number of animals
 - C2* type of animal

2. Generated image
3. Radio buttons: “Animal type matches?”
4. Radio buttons: “Animal count matches?”

Examples




Prompt	Image	Type	Count	Reasoning
3 sheep at the zoo		Yes	No	Not all sheep are clearly identifiable.
3 dogs at the zoo		Yes	Yes	Animal type and count are identifiable.
2 birds in the desert		Yes	No	One bird is cut off by the frame.

Table 3: Example of the Animal Images annotations including a reasoning for the annotation.

Tips

- Focus on intent: small anatomical distortions are acceptable.
- Count only clearly visible, identifiable animals.
- Flag the item if unsure; do not guess.

2.1.3 Cohen’s Kappa

Cohen’s Kappa is a statistical measure used to quantify the agreement between two annotators. It was introduced by Jacob Cohen in 1960 [30]. This measure was developed because percent agreement does not account for chance agreement.

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

The value κ is calculated using the observed agreement p_o and the expected agreement by chance p_e .

- p_o : The proportion of times both annotators agree, calculated as the count of agreements divided by the total number of items.
- p_e : The expected agreement by chance, calculated as the sum of the probabilities of each category agreeing by chance.
- p_{true} : The product of the probability that each annotator labels an item as true.
- p_{false} : The product of the probability that each annotator labels an item as false.

Landis and Koch [31] provided a common interpretation of κ values:

- $k < 0$: No agreement
- $0 \leq k < 0.20$: Slight agreement
- $0.21 \leq k < 0.40$: Fair agreement
- $0.41 \leq k < 0.60$: Moderate agreement
- $0.61 \leq k < 0.80$: Substantial agreement
- $0.81 \leq k \leq 1.00$: Near perfect agreement

These cutoffs are not universally accepted and were not supported by empirical evidence from Landis and Koch. However, they are widely used as a practical guideline to interpret Cohen’s Kappa.

2.2 Evaluate Attribute Control Success

In controlled generation (CG), Attribute Control Success (ACS) measures whether the generated outputs reflect the specified attributes or conditions. Evaluating this success is a critical step in assessing the performance of generative models. Importantly, ACS is not measured on individual generated samples but rather on an entire dataset to estimate the overall effectiveness of control.

Formally, the generation process can be framed as a function:

$$\mathcal{G} : \mathcal{C} \rightarrow \mathcal{O}$$

where $\mathcal{C} = \{c_1, \dots, c_n\}$ denotes the set of controllable attributes (e.g., sentiment, animal type, animal count), and \mathcal{O} is the output domain (such as text, image, or audio). The generator \mathcal{G} maps attributes \mathcal{C} to generated outputs \mathcal{O} .

The evaluation involves a function mapping the generated outputs back to the attribute space:

$$\mathcal{M} : \mathcal{O} \rightarrow \mathcal{C}$$

To evaluate the model, a test set is required:

$$\mathcal{T} = \{(c_n, o_n, h_n, m_n) | n = 1, \dots, N\}$$

Where:

- $c_n \in \mathcal{C}$ is the control attribute for sample n
- $o_n = \mathcal{G}(c_n)$ is the generated output
- $h_n = \mathcal{M}_h(c_n, o_n)$ is the human annotation
- $m_n = \mathcal{M}_{clf}(c_n, o_n)$ is the classifier prediction

If human annotations h_n were available for every item in the test set, evaluating the generative model would be straightforward by treating h_n as ground truth. However, obtaining human annotations at scale is often infeasible due to cost and time constraints. As an alternative, classifier predictions m_n are used for all items. While classifier predictions are inexpensive and yield low-variance estimates, they tend to be biased. In contrast, human annotations, though more accurate and unbiased, typically exhibit higher variance due to annotator subjectivity [11, 12].

Control success is then defined as:

- $y_n = \mathbb{I}[h_n = c_n]$, indicating control success according to human annotation
- $\hat{y}_n = \mathbb{I}[m_n = c_n]$, indicating control success according to the classifier

2.2.1 Beta Distributions

The beta distribution is a family of continuous probability distributions defined on the interval $[0, 1]$, parameterized by two positive shape parameters, α and β . It is commonly used to model random variables that represent proportions or probabilities.

In Bayesian statistics, the beta distribution serves as the conjugate prior for Bernoulli and binomial likelihoods, which means that if the prior over a probability parameter is Beta distributed, the posterior distribution after observing Bernoulli or binomial data is also a Beta distribution. This conjugacy makes Bayesian updating straightforward and computationally efficient.

The shape of the Beta distribution depends on the values of α and β and can represent a wide range of beliefs about the probability parameter from uniform (no prior knowledge) to highly skewed or peaked distributions (strong prior knowledge). A common non-informative prior is the uniform distribution $Beta(1, 1)$, where both α and β equal 1, representing equal belief over the entire interval.

This distribution is particularly suitable for modeling binary outcomes, such as success/failure experiments, where one wants to estimate the underlying success probability and update this estimate as more data becomes available.

2.2.1.1 Example

Suppose we want to estimate the probability p of success in a binary experiment (e.g., flipping a biased coin and getting heads).

We start with a uniform prior belief on p , modeled as:

$$p \sim \text{Beta}(\alpha = 1, \beta = 1)$$

which reflects no prior preference for any value between 0 and 1.

Now, suppose we observe a sequence of trials: out of 10 flips, 7 are heads (successes) and 3 are tails (failures).

Using Bayesian updating, we can compute the posterior distribution for θ by updating the Beta parameters as:

$$\alpha' = \alpha + \text{number of successes} = 1 + 7 = 8$$

$$\beta' = \beta + \text{number of failures} = 1 + 3 = 4$$

The posterior distribution is therefore:

$$p|\text{data} \sim \text{Beta}(8, 4)$$

This posterior reflects our updated belief about the success probability p , incorporating both the prior and the observed data. The mean of this Beta distribution, which can be used as a point estimate of p , is:

$$\mathbb{E}[p] = \frac{\alpha'}{\alpha' + \beta'} = \frac{8}{8 + 4} = \frac{8}{12} = 0.67$$

Hence, after observing the data, our best estimate of the success probability is approximately 0.67, reflecting the observed 7 successes out of 10 trials, while also incorporating prior uncertainty.

2.2.2 Classify & Count

A straightforward method for quantifying the proportion of positive cases in a dataset is the Classify & Count (CC) approach, where the predicted control success for each item, \hat{y}_n , is summed and divided by the total number of items, N :

$$p_{CC} = \frac{1}{N} \sum_{i=1}^N \hat{y}_i$$

In binary classification, this corresponds to counting all positive predictions, which include both true positive (TP) and false positive (FP), and dividing by the total number of samples. As a result, FP can significantly bias the estimated success rate.

Forman [11] showed that under the assumption that the classifier’s behavior on the target data remains consistent with its behavior on the training data (i.e., the conditional probabilities $P(\hat{y}|y)$ do not change), the estimate p_{CC} can be approximated by:

$$p_{CC} \approx TPR \cdot p + FPR \cdot (1 - p)$$

Here, $TPR()$ and $FPR()$ are properties of the classifier, and p is the true, but unknown, success rate in the dataset. This relationship highlights that unless the classifier is perfect ($TPR = 1$ and $FPR = 0$), the CC estimate suffers from systematic bias.

Moreover, the CC method provides only a point estimate without accounting for uncertainty in the prediction. To address this, a Bayesian approach models the success rate as a probability distribution, incorporating prior beliefs and observed data.

Using a Beta prior (e.g., $Beta(1, 1)$ representing a uniform prior), and observing s_{CC} successes (positive predictions), the posterior distribution over the success rate p_{CC} is:

$$s_{CC} = \sum_{i=1}^N \hat{y}_i \tag{1}$$

$$p_{CC} \sim Beta(s_{CC} + 1, N - s_{CC} + 1) \tag{2}$$

2.2.3 Bayesian Classify & Count

In 2022, von Däniken et al. [28] proposed a Bayesian method to quantify classification success by explicitly modeling uncertainties in the classifier’s True Positive Rate (TPR) and False Positive Rate (FPR). Originally developed for automated evaluation of text generation, this approach can be repurposed for quantification tasks in other domains.

The core idea is to treat the TPR and FPR not as fixed values but as random variables modeled by Beta distributions. These distributions are parameterized using counts from the confusion matrix obtained by comparing classifier predictions against human annotated ground truth on a calibration dataset.

2.2.3.1 Confusion Matrix

The confusion matrix summarizes the classifier’s performance on the calibration set and is defined as:

- **tp**: number of true positives (correctly predicted positive)
- **fn**: number of false negatives (missed positives)
- **fp**: number of false positives (incorrectly predicted positive)

- **tn**: number of true negatives (correctly predicted negative)

These counts provide empirical estimates of the classifier’s behavior, which are used to form Beta priors for TPR and FPR.

2.2.3.2 Model Specification

The Bayesian model is defined by the following components:

$$p \sim \text{Beta}(s_h + 1, B - s_h + 1) \quad (3)$$

$$\text{tpr} \sim \text{Beta}(\text{tp} + 1, \text{fn} + 1) \quad (4)$$

$$\text{fpr} \sim \text{Beta}(\text{fp} + 1, \text{tn} + 1) \quad (5)$$

$$s_m \sim \text{Binomial}(N - B, \text{tpr} \cdot p + \text{fpr} \cdot (1 - p)) \quad (6)$$

- p represents the true success rate (proportion of positive cases) in the full dataset. It is modeled as a Beta distribution based on s_h , the number of positive labels in the human annotated calibration subset of size B .
- **tpr** and **fpr** are random variables representing the classifier’s TPR and FPR, respectively. They are modeled as Beta distributions parameterized by confusion matrix counts.
- s_m is the number of positive predictions made by the classifier on the remaining $N - B$ samples outside the calibration set. It follows a Binomial distribution whose success probability is the weighted sum of TPR and FPR, reflecting the classifier’s expected positive prediction rate given p .

The Beta distribution is the conjugate prior of the Binomial distribution, which means that if the likelihood is Binomial and the prior over the success probability is Beta, the posterior distribution is also Beta. This conjugacy allows for tractable Bayesian updating of success probabilities.

Here, the Binomial distribution models the number of observed positive classifier predictions s_m over $N - B$ independent trials, with success probability reflecting the mixture of true and false positive rates weighted by p .

Since the model includes multiple parameters (p , **tpr**, **fpr**) modeled with Beta priors and a Binomial likelihood, the joint posterior distribution does not have a simple closed form. To approximate this posterior and derive credible intervals or point estimates for p , Markov Chain Monte Carlo (MCMC) sampling is employed.

MCMC methods generate samples from the posterior distribution by constructing a Markov chain that has the desired distribution as its equilibrium distribution. Popular algorithms like the Metropolis-Hastings or Hamiltonian Monte Carlo (HMC) efficiently explore the parameter space, allowing estimation of uncertainty in p , **tpr**, and **fpr**.

These samples provide a full Bayesian quantification of the true success rate, incorporating uncertainty from classifier performance and limited calibration data, which is not captured in classical point estimates.

2.3 Evaluation Toolkit

As part of this project, a modular toolkit was developed to support the evaluation of generative models using pretrained classifiers and to enable calibrated quantification of their outputs.

The toolkit is structured into several tasks that are intended to be executed sequentially. These tasks include:

- **Generate:** The output (e.g., text, image, etc.) is produced by the generative model.
- **Annotate:** A local annotation server is launched with the specified interface (e.g., text classification, image classification, etc.), enabling the collection of human annotations, also referred to as oracle labels.
- **Evaluate:** The defined classifiers are applied to the generated outputs in order to produce the metric annotations.
- **Quantify:** A quantification method (e.g., CC, BCC) is executed, combining human and metric annotations to produce calibrated success estimates for the entire dataset.

Although these tasks are designed to be run in sequence, they can be selectively omitted or replaced with custom implementations to suit specific research workflows.

2.3.1 Data Structures

Across the entire toolkit different data structures are used.

Observation

The core data type is called **Observation**, in python it looks like this:

```

1  @dataclass
2  class Observation:
3      id: str                # Unique identifier used to track samples
4      input: str             # Conditioned input for the generative model
5      output: object         # Generated output (text, image, etc.)
6      oracle: Union[int, str] # Human annotation (class label or id)
7      metric: Union[int, str, List[float]] # Metric annotation (class label, id, or logits)

```

All the tasks accept and return a list of **Observation**. Depending on the task different fields are required and in each task at least one field is set.

Ratings

```

1  @dataclass(frozen=True)
2  class Label():
3      id: int
4      name: str

```

```

1  @dataclass(frozen=True)
2  class Ratings():
3      labels: List[Label]
4      observations: List[Observation]

```

2.3.2 Generate

The **generate** task is responsible for invoking a generative model using a specified input configuration. It can be executed in two primary modes: (1) *full generation*, where the input to the generative model is automatically created from a set of labels, label ratios, and a base prompt; and (2) *partial generation*, where a predefined list of prompts is provided using the **Observation** data structure, with the **input** field pre-filled.

2.3.2.1 Configuration

The required configuration depends on two factors: the type of generative model being used and the mode of generation (full or partial). At the time of writing, the following model types are supported by the toolkit:

- **transformer**: Transformer-based models loaded using the HuggingFace (HF) **transformers** library, supporting both local and hosted models.
- **ollama**: Models accessed via HTTP requests to an Ollama server, which can be hosted locally or remotely.
- **diffusion**: Diffusion models loaded using the HF **diffusers** library.
- **flux**: Flux-based models accessed via the **diffusers** library.
- **stable-cascade**: Stable Cascade models also accessed through the **diffusers** library.

Additionally, custom model implementations can be used, provided they inherit from the abstract **Model** class:

```

1  class Model(ABC):
2      @abstractmethod
3      def generate(self, batch: List[Observation]) -> List[Observation]:
4          pass

```

This abstract base class has been intentionally designed with minimal requirements to support diverse modalities in a flexible and model-agnostic manner.

The generation configuration can be defined in YAML format as shown below:

```

1 generate:
2   type: "<MODEL TYPE>"
3   url: "<MODEL URL>"           # Required if type == ollama
4   name: "<PATH | HF ID>"
5   samples: 10_000              # Required if mode == full generation
6   input: "<PATH>"              # Required if mode == partial generation
7   base_prompt: "<BASE PROMPT>"
8   labels:
9     - id: 0
10       name: positive
11       ratio: .35               # Required if mode == full generation
12     - id: 1
13       name: neutral
14       ratio: .3
15     - id: 2
16       name: negative
17       ratio: .35

```

Additional configuration examples can be found in the companion code repository⁴.

2.3.2.2 Input

In *full generation* mode, only a valid configuration file is required, as the input prompts are generated internally.

In *partial generation* mode, a JSON file containing a list of **Observation** items must be provided. Any pre-existing output values in the observations are overwritten by the task.

An example of such input is:

```

1 [
2   {
3     "id": "0000ec39-21a4-43b5-aea9-e2e182aaad55",
4     "input": "Write a very short story with the title: Lemonade",
5     "output": null,
6     "metric": null,
7     "oracle": null
8   }
9 ]

```

2.3.2.3 Output

The task produces a list of **Observation** items where the fields `id`, `input`, and `output` are populated. The output for each item includes the generated response from the model.

⁴<https://github.com/MrF3lix/generative-model-evaluation-toolkit>

A sample output is shown below:

```

1  [
2    {
3      "id": "0000ec39-21a4-43b5-aea9-e2e182aaad55",
4      "input": "Write a short story with the \n title: Lemonade \n sentiment: Positive",
5      "output": "As she sat on the porch, sipping her freshly made lemonade...",
6      "metric": null,
7      "oracle": null
8    }
9  ]

```

2.3.3 Annotate

The **annotate** task is responsible for launching a Prodigy server to facilitate the collection of human (oracle) annotations. Since annotation depends on manual input and typically requires extended periods of time (ranging from several hours to days), it is not intended to be executed as part of an automated end-to-end pipeline. Instead, this task is usually run independently to allow for careful review and discussion of the annotated items.

2.3.3.1 Configuration

The configuration requires the specification of a **type**, which determines the modality of the data (e.g., text, image, or audio) and loads the appropriate annotation interface. The **dataset** name is used internally by Prodigy to store the annotations and is also used to export them later. The **input** field must point to a file containing a list of **Observation** objects, each of which must include the fields **id**, **input**, and **output**. The **labels** field defines the set of valid labels that can be selected during annotation.

```

1  annotate:
2    type: "<MODALITY>"
3    port: 8080
4    dataset: "<DATASET Name>"
5    input: "<PATH>"
6    labels:
7      - id: 0
8        name: positive
9      - id: 1
10       name: neutral
11      - id: 2
12       name: negative

```


2.3.3.2 Requirements

Prodigy is a commercial annotation tool and requires a valid license key. For academic and research purposes, a free license can be requested from the developers⁵.

2.3.3.3 Input

The task expects a valid configuration and a JSON file containing a list of **Observation** entries. If any oracle annotations are already present in the input, they will be overwritten once the new annotations are collected.

2.3.3.4 Output

The annotated data are stored within Prodigy’s internal database under the specified dataset name. These can be exported and converted back into a list of **Observation** entries using a utility script provided by the toolkit. Currently, this process is not automated, as annotation remains a primarily manual process. See Section 2.3.3.5 for current limitations.

2.3.3.5 Current Limitations

During this project, annotation was carried out by two annotators. After the initial annotation round, all items with annotation disagreements were reviewed and discussed, followed by re-annotation to establish a unified oracle annotation for each sample. At present, the annotation task does not support automatic identification or re-annotation of items with annotator disagreement. This limitation is expected to be addressed in future work to streamline the reconciliation process.

2.3.3.6 Evaluate

The **evaluate** task is responsible for applying the configured classifiers to produce automatic metric annotations. One or more classifiers can be specified, and each classifier is used to assign a predicted label to the **metric** field of each **Observation**.

2.3.3.7 Configuration

The configuration format closely resembles that of the **generate** task, as both involve model invocation. The following model types are currently supported for classification:

- **transformer** — models are loaded via the **transformers** library from HF; both local and remote (Hub-based) models are supported.

⁵<https://prodi.gy>

- **ollama** — HTTP requests are sent to a local or remote Ollama server to obtain predictions.
- **yolo** — YOLO-based models are used for object detection tasks.
- **transformer-image** — HF models optimized for image classification or detection.
- **ollama-image** — multi-modal Ollama-compatible models (e.g., LLaVA) that accept images as input.

A sample configuration is shown below:

```

1 classifier:
2   - id: "<MODEL ID>"
3     type: "<MODEL TYPE>"
4     url: "<MODEL URL>"           # Required if type == ollama
5     name: "<PATH | HF ID>"
6     output: "<CLASS | LOGITS>"
7     labels:
8       - id: 0
9         name: positive
10      - id: 1
11        name: neutral
12      - id: 2
13        name: negative

```

2.3.3.8 Input

The task requires a JSON file containing a list of **Observation** objects. Each observation must contain the fields **id**, **input**, and **output**. If an human annotation is already present, it will be preserved and included in the output. Therefore, the **evaluate** task can be executed either before or after the human annotation phase.

An example input is shown below:

```

1 [
2   {
3     "id": "0000ec39-21a4-43b5-aea9-e2e182aad55",
4     "input": "Write a short story with the \n title: Lemonade \n sentiment: Positive",
5     "output": "As she sat on the porch, sipping her freshly made lemonade...",
6     "metric": null,
7     "oracle": "positive"
8   }
9 ]

```

2.3.3.9 Output

A new list of `Observation` objects is returned for each configured classifier. The field `metric` is filled with the label predicted by the classifier. If multiple classifiers are defined, multiple output files (one per classifier) are created.

An example output is shown below:

```

1  [
2    {
3      "id": "0000ec39-21a4-43b5-aea9-e2e182aad55",
4      "input": "Write a short story with the \n title: Lemonade \n sentiment: Positive",
5      "output": "As she sat on the porch, sipping her freshly made lemonade...",
6      "metric": "positive",
7      "oracle": "positive"
8    }
9  ]

```

2.3.4 Quantify

The `quantify` task represents the core contribution of this project. It applies the specified quantification method to estimate class prevalence based on the list of `Observation` instances.

Currently, the following quantification methods are implemented and supported for binary classification tasks:

- **Classify & Count (CC)** — a naive quantification approach used as a baseline for comparison.
- **Bayesian Classify & Count (BCC)** — a quantification method applicable to both binary and multi-class settings. It requires either binary labels or a categorical class assignment for both human and model (metric) annotations.
- **Calibrated Probabilistic Classify & Count (CPCC)** — a calibrated method that directly uses classifier scores (logits) for quantification [11, 13]. This method is incompatible with zero-shot LLM classifiers, as these do not provide classifier scores.

2.3.4.1 Configuration

The configuration is minimal, as the quantification methods rely on the previously collected metric annotations. Note that `CPCC` method requires logits to be present in the metric field.

```

1  quantify:
2    out: "<OUTPUT PATH>"
3    method: "<CC | BCC | CPCC>"

```

2.3.4.2 Input

The task takes as input a list of **Observation** objects with both metric and oracle labels present. A sample input is shown below:

```

1  [
2    {
3      "id": "0000ec39-21a4-43b5-aea9-e2e182aad55",
4      "input": "Write a short story with the \n title: Lemonade \n sentiment: Positive",
5      "output": "As she sat on the porch, sipping her freshly made lemonade...",
6      "metric": "positive",
7      "oracle": "positive"
8    }
9  ]

```

2.3.4.3 Output

For each classifier, a quantification report is generated. The report contains:

- **Observed Human** — beta distribution parameters estimated from the human annotations.
- **Observed Metric** — beta distribution parameters estimated from the raw classifier predictions.
- **Corrected Metric** — calibrated beta distribution parameters after applying the quantification method.

These reports can be used to generate plots such as those in Figure 4, and to compute divergence or agreement metrics between distributions.

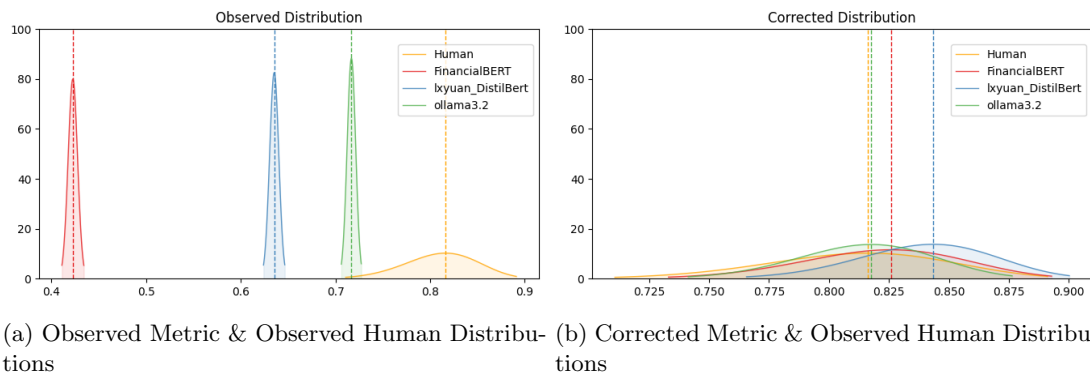


Figure 4: Example output of the quantification step. (a) shows the observed class distributions from classifiers and human annotations before calibration. (b) shows the corrected classifier distributions after calibration, alongside the human distribution.

2.3.4.4 Current Limitations

At the time of writing, all implemented quantification methods support only binary classification tasks. Extending support to multi-class or continuous labels remains future work.

2.4 Experimental Setup

A comprehensive set of experiments was conducted using the previously described toolkit. The goal was to evaluate and calibrate the output of generative models using pretrained classifier models, and to estimate class prevalence through quantification.

Two quantification methods are explored in this project:

- The naive **Classify & Count (CC)** method, which estimates prevalence directly from classifier predictions.
- The **Bayesian Classify & Count (BCC)** method, which calibrates classifier predictions using human annotations to yield corrected class distributions.

For both methods, posterior beta distributions are approximated using the method of moments ⁶.

2.4.0.1 Beta Moment Matching

The quantify task approximates the posterior distribution of success probabilities by computing the empirical mean μ and variance σ^2 from the classifier and human annotations. These estimates are then converted to parameters α and β of the $\text{Beta}(\alpha, \beta)$ distribution using the method of moments:

$$\alpha = \mu \cdot \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right) \quad \beta = (1-\mu) \cdot \left(\frac{\mu(1-\mu)}{\sigma^2} - 1 \right)$$

The empirical values μ and σ^2 are estimated through sampling using a Markov Chain Monte Carlo (MCMC) procedure integrated in the toolkit.

2.4.0.2 Experiment Pairs

Two types of experimental comparisons are performed using these beta-distributed posterior estimates:

1. **Classifier Comparison (Fixed Generator)**: For a fixed generative model, the posterior distributions of different classifiers are compared. Consistency between classifier distributions and the human-annotated distribution indicates unbiased classifier behavior. In contrast, divergence among classifier distributions signals potential bias or instability in classifier performance.

⁶<https://statproofbook.github.io/P/beta-mome.html>

2. **Generator Comparison (Fixed Classifier):** For a fixed classifier, each generative model’s output is compared in a pairwise manner. The resulting beta distributions are used to estimate the probability that one model outperforms another. If consistent rankings of generators emerge across multiple classifiers, the quantification method is considered reliable and robust.

2.4.0.3 Experiment Overview

The experiments are conducted across two modalities: text and image. For each modality, three generative models are evaluated using three different classifiers, resulting in a total of 18 evaluation pipelines per modality. Table 4 provides an overview of all generator-classifier combinations used in the study.

Modality	Generative Model	Classifier Model
Text	LLama3.3 70B	FinancialBERT
		DistilledSentimentStudent
		LLama3.2 3B
	LLama2 7B	FinancialBERT
		DistilledSentimentStudent
		LLama3.2 3B
	Mistral 7B	FinancialBERT
		DistilledSentimentStudent
		LLama3.2 3B
Image	Stable Diffusion 3.5	YOLOv8
		DETR
		LLaVA
	Stable Cascade	YOLOv8
		DETR
		LLaVA
	FLUX.1-dev	YOLOv8
		DETR
		LLaVA

Table 4: Overview of the experiments conducted in this project. For each modality (Text and Image), three generative models are evaluated using three distinct classifier models.

2.4.0.4 Analysis Goals

This experimental setup enables two key types of analysis:

- Assessing how much bias or disagreement exists between classifier models when used to evaluate the same generative output.

- Estimating the relative quality of different generative models, corrected through Bayesian quantification, across diverse classifiers and modalities.

Together, these experiments aim to answer whether reliable performance estimates can be recovered through calibrated quantification, and whether classifier-based evaluation is consistent enough to support fair comparison of generative models.

2.4.1 Consistency

In an ideal scenario, a ground truth containing conditional attributes would be available to directly evaluate the performance of generative models. However, obtaining such comprehensive annotations across an entire dataset is typically infeasible in practice. As a result, this project focuses on deriving consistent performance estimates in the absence of full ground truth.

To address this challenge, the evaluation methodology relies on *consistency measurements* as a proxy for absolute correctness.

The following assumptions are made to guide consistency analysis:

1. The posterior distributions produced by the Bayesian Classify & Count (BCC) method are expected to be closer to the distribution derived from human (oracle) annotations than those produced by the naive Classify & Count (CC) method.
2. The posterior distributions from BCC, when applied to the same generative model but different classifiers, should also be more similar to each other. This would indicate stability and robustness across classifier models.

To quantify the similarity between distributions, the Jensen-Shannon (JS) divergence is used. This metric provides a bounded and symmetric measure of divergence between two probability distributions. A JS value of 0 indicates identical distributions, whereas a value of 1 indicates complete divergence (i.e., no overlap).

In all cases, the JS divergence is measured between the metric posterior and the oracle (human) posterior distribution.

Additionally, the variance of each posterior distribution is computed. A lower variance indicates a narrower credibility band and reflects a higher degree of certainty in the estimated prevalence. In the context of model evaluation, distributions with lower variance and low JS divergence from the human distribution are considered more reliable.

2.4.2 Measurability

To compare whether one generative model $p^{(i)}$ performs better than another $p^{(j)}$, the posterior distributions from a single classifier can be evaluated probabilistically. This is achieved by expressing the comparison as follows:

$$p(p^{(i)} > p^{(j)}) = \Pr[X > Y] \quad (7)$$

$$X \sim \text{Beta}(\alpha_i, \beta_i) \quad (8)$$

$$Y \sim \text{Beta}(\alpha_j, \beta_j) \quad (9)$$

This probability can be estimated using Markov Chain Monte Carlo (MCMC) sampling. Specifically, S independent and identically distributed samples $\{x_s, y_s\}_{s=1}^S$ are drawn from the two beta distributions, and the following estimate is computed:

$$\frac{1}{S} \sum_{s=1}^S \mathbb{I}[x_s > y_s]$$

This yields the probability that the posterior distribution of model $p^{(i)}$ dominates that of $p^{(j)}$ according to the classifier under consideration.

2.4.3 Sample Value

An important consideration in the context of weak supervision is the value of a single metric annotation relative to a human annotation. In situations where only a small number of human annotations are available but a large number of metric annotations exist, the following approach is proposed to quantify the added value of the latter.

The posterior distribution parameters estimated during quantification yield a sum $\hat{B} = \alpha + \beta$, which can be interpreted as the effective sample size. The difference $\hat{B} - B$ represents the number of additional samples effectively contributed by the metric annotations, where B denotes the number of actual human (oracle) annotations.

This difference is defined as the effective additional sample (EAS) size. To calculate the relative contribution of each metric annotation, the EAS is normalized by the number of metric-only annotations ($N - B$), where N is the total number of annotations (both human and metric).

This yields the **sample value** V :

$$V = \frac{\hat{B} - B}{N - B}$$

A sample value of $V = 1$ indicates that a metric annotation contributes as much information as a human annotation. Conversely, a value of $V = 0$ implies that the metric annotation contributes no additional information to the posterior.

3 Data

As part of this project, two datasets spanning two different modalities were created. A fraction of each dataset was annotated by two human annotators to serve as a reference for evaluating quantification performance.

3.1 Sentiment Stories

The first dataset, named *Sentiment Stories*, is a text-based dataset consisting of short stories generated by large language models (LLM). Each story is conditioned on a title $t \in T$ and a target sentiment $s \in \{\text{positive, neutral, negative}\}$, and generated using the following prompt:

```

1 Write a very short story with the title: "<t>"
2 The story should have a "<s>" sentiment.
3 The story should be shorter than 5 sentences.
```

The set of titles T consists of 10'000 unique story titles drawn from the ROC Stories corpus [32]. A total of 10'000 prompts were created and executed using three different generative models to generate 30'000 stories.

3.1.1 Generators

Three open-weight LLMs were selected to generate the stories. All models were run locally to ensure full control over the generation process:

1. **L370B** — LLama-3.3-70B-Instruct [33]
2. **L27B** — LLama-2-7B [34]
3. **Mi7B** — Mistral-7B [35]

3.1.2 Classifiers

Three pretrained sentiment classifiers were selected from HuggingFace (HF) to evaluate the generated stories. These classifiers were used without additional fine-tuning:

1. **FIB** — FinancialBERT [36], a BERT-based model pre-trained on financial texts and fine-tuned on the Financial PhraseBank dataset [37].
2. **DSS** — DistilledSentimentStudent, a DistilBERT model distilled from multilingual sentiment corpora^{7,8}

⁷<https://huggingface.co/datasets/tyqiangz/multilingual-sentiments>

⁸<https://huggingface.co/lxyuan/distilbert-base-multilingual-cased-sentiments-student>

3. **LL3** — LLama3.2-3B [33], used as a prompt-based zero-shot sentiment classifier.

3.1.3 Human Annotations

For each generator, a subset of $B = 100$ stories was independently annotated by two PhD-level computational linguistics researchers. The annotated subset was shared across all three generators, resulting in a total of 300 annotated stories derived from 100 distinct prompts.

Two variables were annotated:

1. *Title match* — whether the story meaningfully reflects the provided title.
2. *Sentiment* — the perceived sentiment of the story.

Inter-annotator agreement was measured using Cohen’s kappa (κ). For the title match variable, a slight agreement was observed ($\kappa = 0.05$). In contrast, sentiment annotations showed substantial agreement ($\kappa = 0.76$).

A second annotation round was conducted to resolve disagreements. Final labels were agreed upon via discussion. Any story failing the title match criterion was excluded from further analysis.

3.1.4 Corpus Statistics

The final corpus contains $N = 10'000$ samples per generator. The sentiment conditioning was distributed as follows: 3'500 positive, 3'500 negative, and 3'000 neutral prompts. For each generator, $B = 100$ samples were annotated by humans, and all N samples were classified by each of the three classifiers.

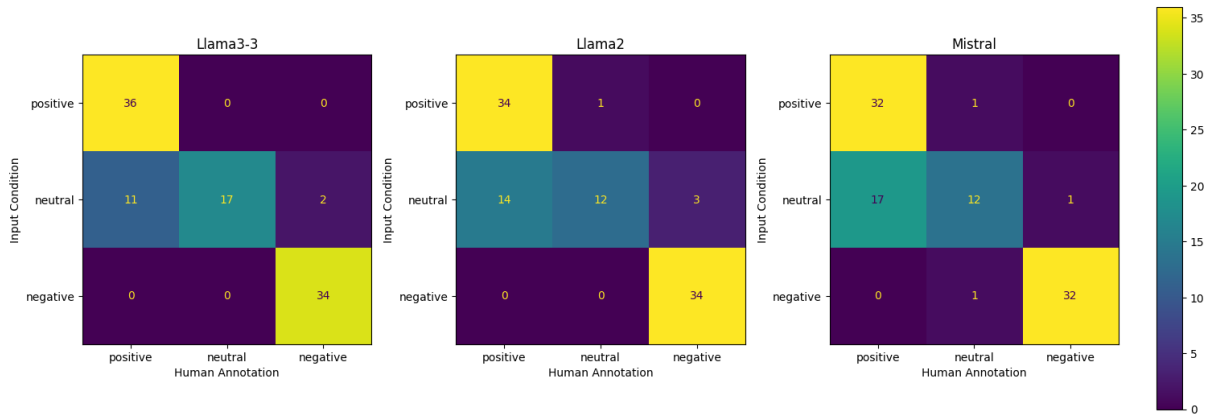


Figure 5: Confusion matrices comparing the input sentiment condition and the human-annotated sentiment.

Figure 5 presents confusion matrices that compare the intended sentiment (as specified in the generation prompt) with the human-annotated sentiment. A consistent trend can be observed across all models:

neutral prompts are frequently annotated as positive. This is especially pronounced for *Mistral-7B*, where 17 of 30 neutral samples were labeled as positive. For the other sentiment categories, mismatches are rare (typically 1–2 cases), indicating that the models generally follow the conditioning prompt but exhibit a slight bias toward generating positive content.

All annotations were binarized into two variables: *sentiment match/mismatch* and *title match/mismatch*.

Generator	Title match (%)	Sentiment match (%)
Llama-3.3-70B	100	87
Llama-2-7B	98	80
Mistral-7B	96	76

Table 5: Percentage of generated stories where the title and sentiment match the human annotation.

Table 5 summarizes the proportion of title and sentiment matches per generator. Most stories match their assigned titles, confirming the models’ capability to respect this constraint. Only stories with matching titles are retained for subsequent experiments. The sentiment match percentages serve as an early indicator of classifier difficulty.

	FIB	DSS	LL3
Macro F1	0.275	0.521	0.671
TPR _{pos}	0.216	0.695	1.000
FPR _{pos}	0.074	0.127	0.195
TPR _{neu}	0.886	0.045	0.089
FPR _{neu}	0.784	0.000	0.000
TPR _{neg}	0.129	0.943	0.990
FPR _{neg}	0.016	0.388	0.063
TPR _{match}	0.327	0.696	0.844
FPR _{match}	0.913	0.0	0.052

Table 6: Classifier performance on the human-annotated subset of 300 stories.

Table 6 reports the classification performance of each model against the human annotations using standard metrics (macro F1, TPR, FPR). The results show notable variation:

- **FIB** achieves the lowest macro F1 score (0.275), with poor True Positive Rate (TPR) for both positive and negative classes, and a tendency to overpredict the neutral class—evident in the elevated False Positive Rate (FPR) for neutral.
- **DSS** shows the opposite behavior, predominantly classifying stories as positive or negative while largely ignoring the neutral class.
- **LL3** performs best overall, with a macro F1 of 0.67. It achieves high TPR for positive and negative labels and maintains relatively low FPR, although it consistently underpredicts the neutral category.

3.2 Animal Images

The second dataset created for this project is the *Animal Images* dataset. It is a image dataset comprising image-generation prompts, the resulting images, human annotations, and classifier evaluations.

To generate the images, 10'000 unique prompts were composed using a combination of five conditional attributes. Two of these attributes, *animal type* and *number of animals*, were later used for evaluation. The full set of conditional dimensions includes:

- **Number:** [1, 2, 3, 4, 5, 6]
- **Animal:** ['bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe']
- **Place:** ['in the arctic', 'on a mountain', 'in the desert', 'at the zoo', 'in the forest', 'in a city', 'on the lake', 'at home']
- **Lighting:** ['during the day', 'in the evening', 'at night', 'during sunrise', 'at noon']
- **Weather:** ['while it is raining', 'while snow is falling', 'with the sun shining', 'in a snow storm', 'in a thunderstorm']

Each prompt followed the same textual template:

```
1 <NUMBER> <ANIMAL> <PLACE> <LIGHTING> <WEATHER>. Generate a hyper realistic image.
```

This process yielded a total of 30'000 images using three different generative models. Each model received the same set of 10'000 prompts, allowing for consistent evaluation. From this set, 100 prompts were randomly sampled and annotated by two independent annotators for each generator, resulting in a total of 300 human-labeled images.

3.2.1 Generators

Three pretrained image generation models were used without modification:

- **SD35:** Stable Diffusion 3.5 [38]⁹
- **StCa:** Stable Cascade [39]¹⁰
- **FLX1:** FLUX.1-dev [40]¹¹

⁹<https://huggingface.co/stabilityai/stable-diffusion-3.5-medium>

¹⁰<https://huggingface.co/stabilityai/stable-cascade>

¹¹<https://huggingface.co/black-forest-labs/FLUX.1-dev>

3.2.2 Classifiers

Two attributes were automatically evaluated for each image: (1) the animal type present, and (2) the number of animals shown. Three pretrained object detection models were used to extract this information. All models were used without fine-tuning:

- **YLO**: YOLOv8 [15], a fast and lightweight real-time object detector using its detection head for class labels and bounding boxes.
- **LLV**: LLaVA [41], a vision–language model capable of instruction following. It was prompted with: “*What animal is shown and how many are there?*”
- **DTR**: DETR [16], a transformer-based object detector robust to overlapping instances and complex spatial layouts.

An output was marked correct if the classifier detected the correct animal type and matched the exact number of instances via bounding boxes.

3.2.3 Human Annotations

For each generator, a subset of $B = 100$ images was annotated by two PhD-level computational linguistics researchers. The same 100 prompts were used across all three generators, resulting in 300 annotated images.

Two evaluation variables were annotated:

1. **Animal type**: whether the correct animal type appeared in the image.
2. **Animal count**: whether the correct number of animals of the specified type appeared.

Cohen’s kappa (κ) was computed to assess inter-annotator agreement. For animal type, substantial agreement was observed ($\kappa = 0.94$). For animal count, substantial agreement was also found ($\kappa = 0.82$).

3.2.4 Corpus Statistics

The final corpus contains $N = 10'000$ images per generator, with animal types equally distributed (1'000 images per type). For each generator, $B = 100$ samples were annotated by humans, and all images were automatically classified by the object detectors.

Table 7 shows the human-assessed success rates for both animal type and count. *Stable Diffusion 3.5* and *FLUX-1.dev* both achieved perfect accuracy on animal type. Their animal count accuracy was 61% and 65%, respectively. *Stable Cascade* lagged significantly on animal count, with only 5% accuracy, though its type accuracy remained high at 97%.

Table 8 reports classifier performance on the human-annotated subset. The *LLV* model achieved the lowest macro F1 score (0.551) and true positive rate ($\text{TPR} = 0.552$), suggesting difficulty in jointly

Generator	Type match (%)	Count match (%)
Stable Diffusion 3.5	100	61
Stable Cascade	72	5
Black Forest FLUX-1	100	65

Table 7: Percentage of generated images in which the requested animal type and count are correctly realized.

	LVV	YLO	DTR
Macro F1	0.551	0.869	0.927
TPR _{count match}	0.552	0.882	0.93
FPR _{count match}	0.107	0.14	0.104

Table 8: Performance of image classifiers on the human-annotated subset (macro F1, TPR, FPR).

identifying animal type and count. In contrast, *YOLOv8* and *DETR* performed strongly, each achieving a macro F1 greater than 0.85, with high TPR and low FPR.

4 Results

For both created datasets, the metric annotations were used alongside human annotations to compute quantification results using the Classify & Count (CC) and Bayesian Classify & Count (BCC) methods. These methods were applied to each classifier, and evaluated in terms of three criteria: consistency, measurability, and sample value.

4.1 Consistency

To assess consistency, the CC and BCC methods were applied to every generator–classifier pair. The Jensen-Shannon (JS) divergence between the resulting posterior distributions and the human annotation distribution was computed, along with the variance of the posterior distributions.

Generator	Classifier	CC		BCC	
		JS	Variance	JS	Variance
L370B	FIB	1.0000	2.49×10^{-5}	0.1620	1.14×10^{-3}
	DSS	1.0000	2.38×10^{-5}	0.0726	9.04×10^{-4}
	LL3	0.9982	1.97×10^{-5}	0.1239	7.52×10^{-4}
L27B	FIB	1.0000	2.49×10^{-5}	0.1305	1.19×10^{-3}
	DSS	0.9994	2.33×10^{-5}	0.3561	8.44×10^{-4}
	LL3	0.9648	2.03×10^{-5}	0.1686	8.53×10^{-4}
Mi7B	FIB	1.0000	2.30×10^{-5}	0.1255	1.25×10^{-3}
	DSS	0.9995	2.42×10^{-5}	0.2160	1.02×10^{-3}
	LL3	0.9399	2.09×10^{-5}	0.1942	9.69×10^{-4}

Table 9: Consistency metrics per generator–classifier pair for the sentiment story dataset. For each quantification method (CC & BCC), the JS divergence to the human posterior (lower is better) and the posterior variance (lower is better) are reported.

Table 9 presents the consistency metrics for the sentiment story dataset across all generator–classifier pairs. Results show that the CC method exhibits consistently high JS divergence values, indicating substantial misalignment between the classifier-derived posteriors and the human-annotated distribution. Additionally, the CC method yields very low variance across all settings.

In contrast, the BCC method demonstrates substantially lower JS divergence, with the best result observed for the Llama-3.3-70B and DSS classifier pair ($JS = 0.0726$), and the worst for the Llama-2.7B and DSS pair ($JS = 0.3561$). Notably, the posterior variance under BCC is one to two orders of magnitude higher than under CC, aligning more closely with the variance observed in the human distribution.

Figure 6 supports these findings visually. The CC distributions are highly concentrated (low variance) but misaligned with the human distribution, leading to high JS divergence. In contrast, BCC posteriors cluster closely around the human distribution and display comparable variance. Among generators, *Llama-3.3-70B* consistently produces distributions most similar to the human annotation. The plot also highlights classifier biases, which BCC effectively corrects.

Table 10 summarizes the results for the animal image dataset, evaluated separately for animal count

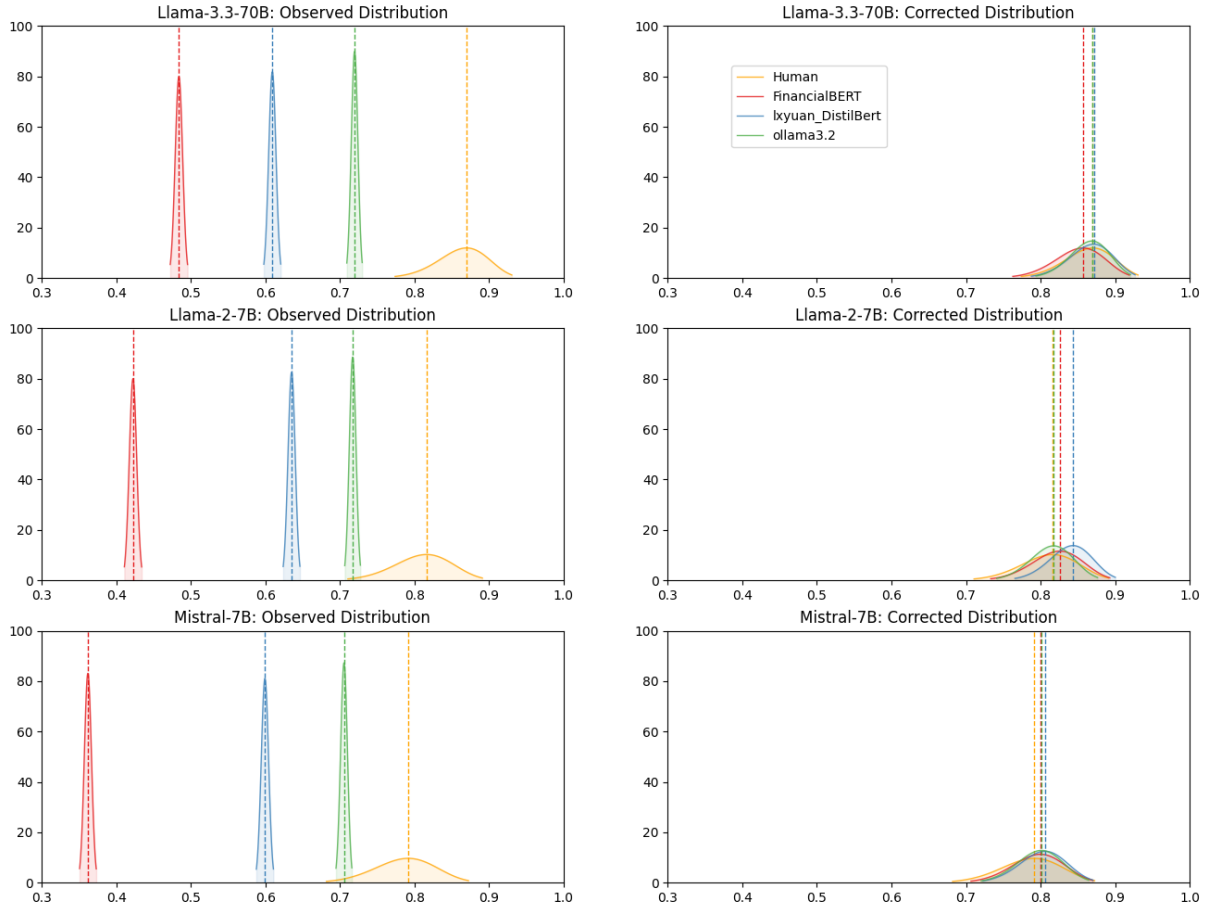


Figure 6: Observed distributions from the CC method compared to corrected distributions from the BCC posterior for each generator.

Task	Generator	Classifiers	CC		BCC	
			JS	Variance	JS	Variance
Animal Count	SD35	YLO	0.7897	2.37×10^{-5}	0.3455	1.34×10^{-3}
		DTR	0.7992	2.35×10^{-5}	0.2350	1.02×10^{-3}
		LLV	0.8374	2.46×10^{-5}	0.0996	1.67×10^{-3}
	StCa	YLO	0.9225	9.17×10^{-6}	0.4954	4.58×10^{-4}
		DTR	0.8935	8.61×10^{-6}	0.5981	3.79×10^{-4}
		LLV	1.0000	2.00×10^{-8}	0.1692	4.70×10^{-4}
	FLX1	YLO	0.8789	2.45×10^{-5}	0.2129	1.06×10^{-3}
		DTR	0.7973	2.31×10^{-5}	0.3475	6.19×10^{-4}
		LLV	0.8374	2.39×10^{-5}	0.2308	1.06×10^{-3}
Animal Type	SD35	YLO	0.7807	1.16×10^{-7}	0.3922	2.48×10^{-5}
		DTR	0.8372	5.65×10^{-8}	0.4581	2.21×10^{-5}
		LLV	1.0000	1.61×10^{-5}	0.0181	8.60×10^{-5}
	StCa	YLO	0.8425	2.21×10^{-5}	0.2046	1.09×10^{-3}
		DTR	0.8580	2.24×10^{-5}	0.2829	7.14×10^{-4}
		LLV	1.0000	1.99×10^{-8}	0.0400	2.00×10^{-3}
	FLX1	YLO	0.9354	2.90×10^{-6}	0.1275	5.02×10^{-5}
		DTR	0.8280	1.34×10^{-6}	0.2220	4.51×10^{-5}
		LLV	1.0000	1.70×10^{-5}	0.0277	8.12×10^{-5}

Table 10: Consistency metrics per generator–classifier pair for the animal image dataset. Each row reports the Jensen-Shannon (JS) and posterior variance for both the animal count and animal type tasks using CC and BCC methods.

and animal type. As with the sentiment dataset, CC produces high JS divergence (often close to 1), especially for more challenging settings. Among the generators, *SD35* yields the lowest divergence across classifiers for both tasks, with JS values below 0.8. Among classifiers, *YLO* consistently achieves the lowest divergence to the human distribution.

The posterior variance for CC remains low across all settings; the *SD35*–*DTR* pair has the lowest recorded variance (5.65×10^{-8}). Under BCC, the JS divergence drops, ranging from 0.0181 to 0.5981. The only notable outlier is the *StCa*–*DTR* pair, which continues to exhibit high divergence in the animal count task, even after Bayesian correction.

These results confirm the general trend: CC yields highly concentrated but biased estimates, whereas BCC improves alignment with human distributions at the cost of increased—but more realistic—variance.

Figure 7 shows that although CC distributions are not as widely spread as their JS divergence values might suggest, they still deviate notably from the human distribution. The BCC corrected distributions more closely align with the human baseline. This figure also explains the poor performance of the *StCa*–*DTR* combination: the underlying generator appears incapable of rendering the correct number of animals, as corroborated by Table 7, where only 5% of its outputs meet the count specification.

Figure 8 reveals that the *SD35* and *FLX1* generators produce highly accurate animal types. Among classifiers, only *LLV* deviates significantly when using the CC method. However, after applying BCC,

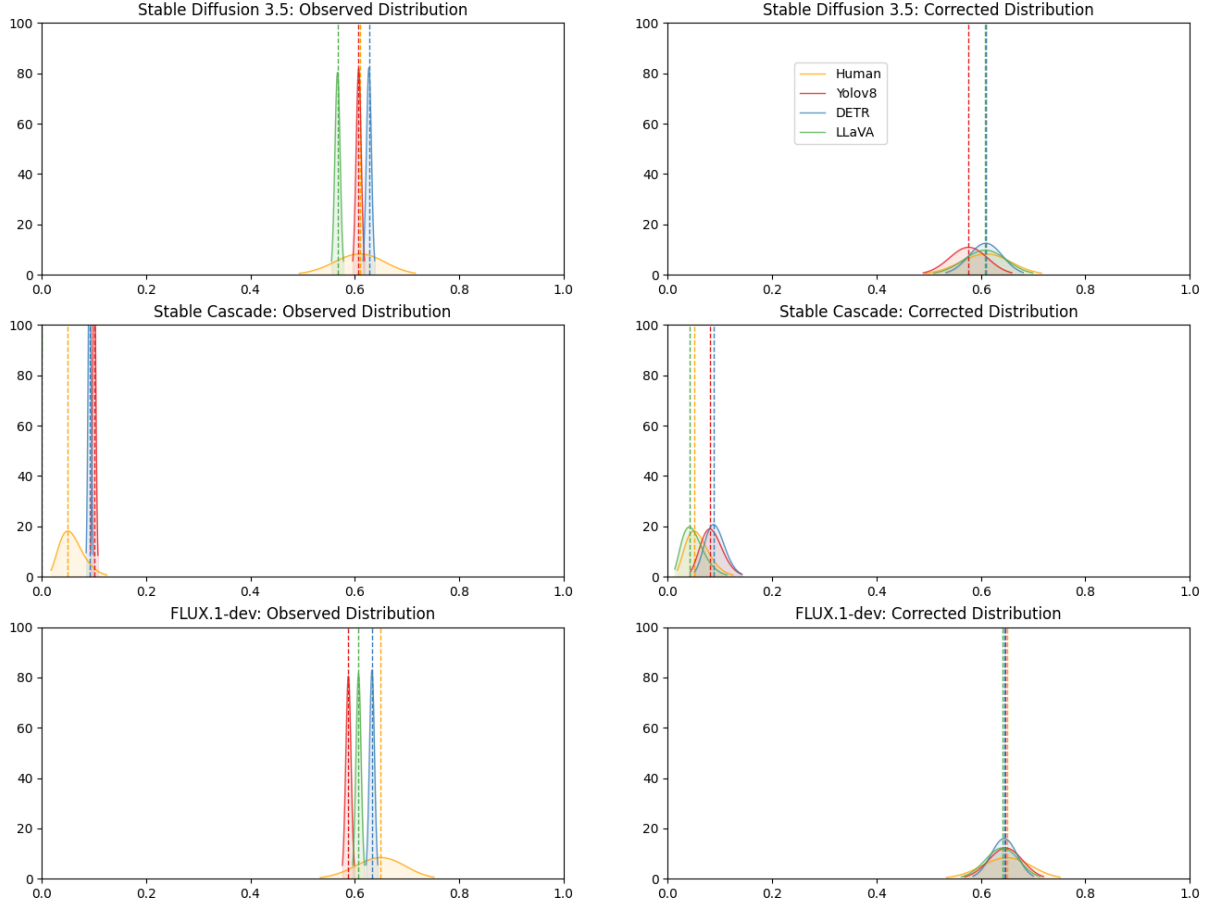


Figure 7: Observed distributions from the CC method versus corrected BCC posteriors for the animal count task across all generators.

the corrected posterior aligns well with both the human annotation and the distributions from the other classifiers. As in other tasks, *StCa* performs worse than the other generators.

4.2 Measurability

To compare whether one generator is better than another, the posterior distributions from each classifier are compared across generators. This can be used to rank the generators according to each classifier. If all classifiers show the same ranking, it means the method produces consistent performance estimates.

Table 11 shows that with the CC method, not all classifiers agree on the ranking of generators. For example, the *FIB* and *LL3* classifiers give a high probability that the *L370B* generator is better than the *L27B* generator. However, the *DSS* classifier ranks *L27B* higher than *L370B*.

For the other comparisons, the CC method gives consistent results across classifiers. The biggest difference

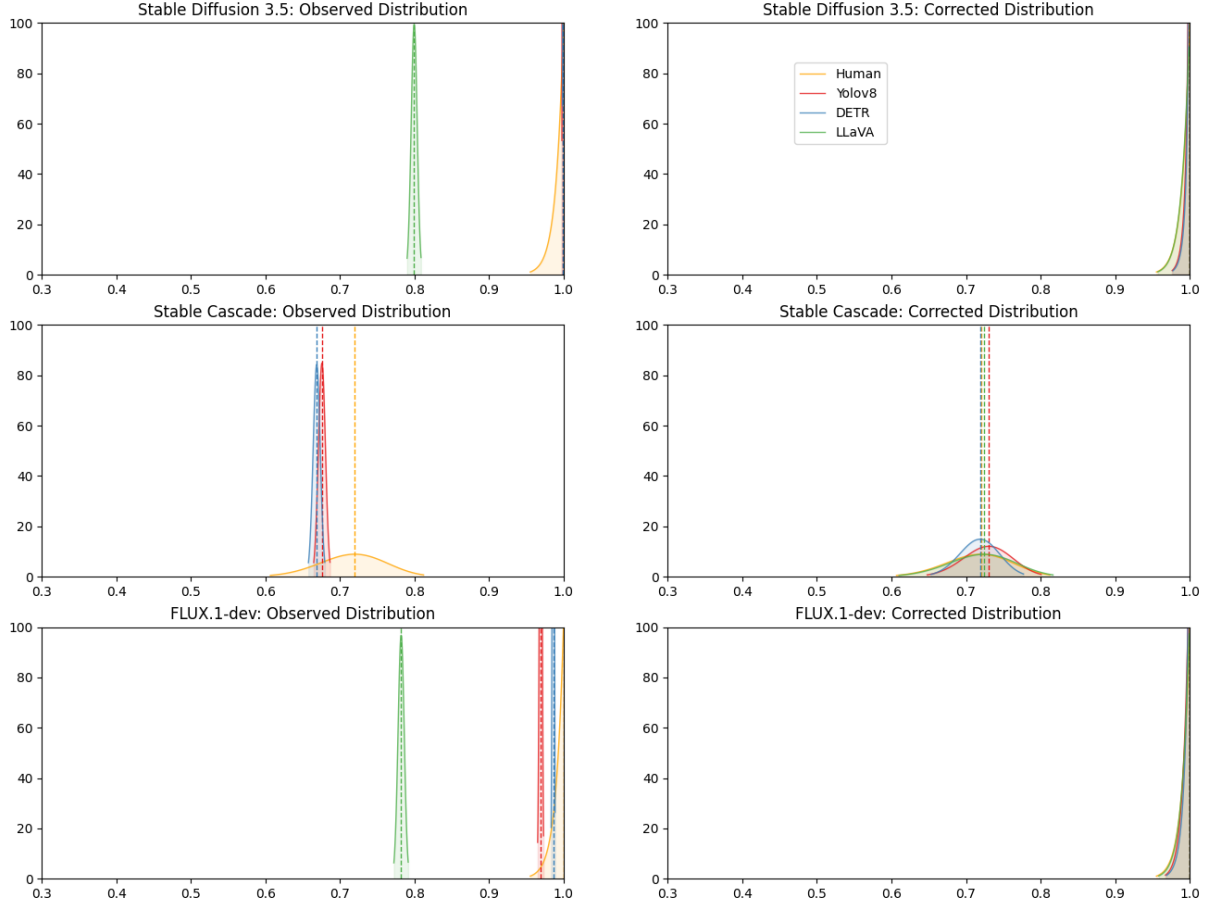


Figure 8: Observed distributions from the CC method versus corrected BCC posteriors for the animal type task across all generators.

Method	Classifier	$P(L3\ 70B > L2\ 7B)$	$P(L3\ 70B > Mi7B)$	$P(L2\ 7B > Mi7B)$
Human	—	0.85	0.93	0.66
CC	FIB	1.00	1.00	1.00
	DSS	0.00	0.92	1.00
	LL3	0.67	0.99	0.96
BCC	FIB	0.73	0.87	0.70
	DSS	0.74	0.93	0.81
	LL3	0.89	0.95	0.65

Table 11: Posterior probability that one generator outperforms another, estimated from Beta posteriors. The Human-only rows provide the reference; CC and BCC show uncalibrated and calibrated metric estimates, respectively.

compared to the human posterior is in the $L27B$ vs. $Mi7B$ comparison. The classifiers using CC give

a very high probability that *L27B* is better, while the human posterior shows a probability close to 0.5, suggesting that the two models are about equal.

The BCC method gives much more consistent results. The probabilities are closely aligned with the human posterior across all generator comparisons.

Method	Classifier	$P(\text{SD35} > \text{StCa})$	$P(\text{SD35} > \text{FLX1})$	$P(\text{StCa} > \text{FLX1})$
Human	–	1.0	0.49843	0.0
CC	YLO	1.0	1.0	0.0
	DTR	1.0	1.0	0.0
	LLV	1.0	0.99868	0.0
BCC	YLO	1.0	0.71162	0.0
	DTR	1.0	0.6917	0.0
	LLV	1.0	0.48717	0.0

Table 12: Posterior probability that one generator outperforms another, estimated from Beta posteriors on the animal count task. The Human-only rows provide the reference; CC and BCC show uncalibrated and calibrated metric estimates, respectively.

Table 12 shows that the CC method gives a clear and confident ranking across all classifiers. This ranking is mostly in line with the human posterior. The main exception is the comparison between *SD35* and *FLX1*: the CC method gives a high probability that *SD35* is better, but the human posterior shows that *FLX1* performs slightly better.

The results from the method correct this mismatch. The probability that *SD35* is better than *FLX1* is now much closer to the human judgment.

Method	Classifier	$P(\text{SD35} > \text{StCa})$	$P(\text{SD35} > \text{FLX1})$	$P(\text{StCa} > \text{FLX1})$
Human	–	1.0	0.27823	0.0
CC	YLO	1.0	0.99756	0.0
	DTR	1.0	0.19338	0.0
	LLV	1.0	1.0	0.0
BCC	YLO	1.0	0.07458	0.0
	DTR	1.0	0.19404	0.0
	LLV	1.0	0.26272	0.0

Table 13: Posterior probability that one generator outperforms another, estimated from Beta posteriors on the animal type task. The Human-only rows provide the reference; CC and BCC show uncalibrated and calibrated metric estimates, respectively.

Table 13 shows that the CC method leads to inconsistent rankings between classifiers for the comparison between *SD35* and *FLX1*. The *DTR* classifier favors *FLX1*, which matches the human posterior, while the other two classifiers rank *SD35* higher.

Using the BCC method, the results become much more consistent. The corrected posteriors are close to the human posterior in all comparisons.

4.3 Sample Value

The sample value measures how much information a metric annotation provides compared to a human annotation.

Generator	Sentiment Stories					
	FIB		DSS		LL3	
	V	EAS	V	EAS	V	EAS
L370B	1.04×10^{-3}	10.28	2.81×10^{-3}	27.81	5.66×10^{-3}	56.05
L27B	2.27×10^{-3}	22.44	5.94×10^{-3}	58.81	7.80×10^{-3}	77.19
Mi7B	2.98×10^{-3}	29.52	5.47×10^{-3}	54.16	6.44×10^{-3}	63.73

Generator	Animal Images					
	YLO		DTR		LLV	
	V	EAS	V	EAS	V	EAS
SD35	6.93×10^{-3}	68.65	1.13×10^{-2}	111.46	3.94×10^{-3}	39.05
StCa	1.46×10^{-3}	14.48	8.33×10^{-3}	82.43	-3.30×10^{-3}	-32.63
FLX1	1.15×10^{-2}	113.89	2.72×10^{-2}	269.5	1.18×10^{-2}	117.24

Table 14: Per-sample human-equivalent value V for each generator–classifier pair. Positive values indicate that a metric score adds information; negative values indicate net noise.

Table 14 shows that even the best metric, *LL3*, adds the equivalent of only 77.19 human annotations. This is still less than the 100 human labels used in the calibration set. Weaker metrics contribute between 10 and 60 human-equivalent labels.

This means that while the BCC estimates are accurate, the added value of each metric sample is still small. Thousands of metric predictions only replace a few dozen human judgments.

A similar result is seen for the animal image dataset. The best classifier adds about 269 human-equivalent annotations. This is more than in the language task, but still far from replacing human labels.

5 Conclusion

This project explored how conditional generation can be evaluated using pre-trained classifiers. It was shown that the naive Classify & Count (CC) approach often leads to biased and inconsistent estimates of generator performance. To address this, a Bayesian calibration method, called Bayesian Classify & Count (BCC), was implemented. This method uses a small number of human annotations to model the uncertainty that comes with relying on pre-trained classifiers, resulting in more consistent and less biased estimates.

A toolkit was developed that implements this method and can be applied across different data modalities, such as text and images. In addition, a new dataset was created to support this evaluation setup, containing prompts, generated outputs, human annotations, and classifier-based metric annotations.

Experiments demonstrated that the BCC method produces more robust and reliable estimates of conditional generation performance, compared to the naive CC method.

5.1 Future Work

To extend the usefulness of the toolkit in more realistic and complex settings, future work could add support for multi-class classification and allow quantification methods to make use of classifier logits. This would enable the integration of alternative calibration methods such as Probabilistic Classify & Count (PCC) and Calibrated Probabilistic Classify & Count (CPCC). Furthermore, the toolkit could be expanded to support evaluation setups involving additional sources of uncertainty, making it suitable for broader and more challenging applications.

List of Figures

1	Motivating Example. Letting a generator write 10'000 stories controlled by a sentiment label, and then letting sentiment classifiers check if the sentiment was correctly expressed. The automated ratings are then compared to 100 human judgments.	1
2	Overview of the annotation tool Prodigy. Showing a generated story including the title and the labels to set by the annotator. Two variables are annotated in this scenario, the sentiment (Positive, Neutral, Negative), and whether the title matches the story or not. .	4
3	Overview of the annotation tool Prodigy. Showing a generated image including the condition and the labels to set by the annotator. Two variables are annotated in this scenario, whether the animal type matches the condition and whether the animal count matches the condition.	5
4	Example output of the quantification step. (a) shows the observed class distributions from classifiers and human annotations before calibration. (b) shows the corrected classifier distributions after calibration, alongside the human distribution.	21
5	Confusion matrices comparing the input sentiment condition and the human-annotated sentiment.	27
6	Observed distributions from the CC method compared to corrected distributions from the BCC posterior for each generator.	33
7	Observed distributions from the CC method versus corrected BCC posteriors for the animal count task across all generators.	35
8	Observed distributions from the CC method versus corrected BCC posteriors for the animal type task across all generators.	36

List of Tables

1	Sentiment labels	6
2	Illustrative annotation examples.	6
3	Example of the Animal Images annotations including a reasoning for the annotation. . . .	8
4	Overview of the experiments conducted in this project. For each modality (Text and Image), three generative models are evaluated using three distinct classifier models. . . .	23
5	Percentage of generated stories where the title and sentiment match the human annotation.	28
6	Classifier performance on the human-annotated subset of 300 stories.	28
7	Percentage of generated images in which the requested animal type and count are correctly realized.	31

8	Performance of image classifiers on the human-annotated subset (macro F1, TPR, FPR).	31
9	Consistency metrics per generator–classifier pair for the sentiment story dataset. For each quantification method (CC & BCC), the JS divergence to the human posterior (lower is better) and the posterior variance (lower is better) are reported.	32
10	Consistency metrics per generator–classifier pair for the animal image dataset. Each row reports the Jensen-Shannon (JS) and posterior variance for both the animal count and animal type tasks using CC and BCC methods.	34
11	Posterior probability that one generator outperforms another, estimated from Beta posteriors. The Human-only rows provide the reference; CC and BCC show uncalibrated and calibrated metric estimates, respectively.	36
12	Posterior probability that one generator outperforms another, estimated from Beta posteriors on the animal count task. The Human-only rows provide the reference; CC and BCC show uncalibrated and calibrated metric estimates, respectively.	37
13	Posterior probability that one generator outperforms another, estimated from Beta posteriors on the animal type task. The Human-only rows provide the reference; CC and BCC show uncalibrated and calibrated metric estimates, respectively.	37
14	Per-sample human-equivalent value V for each generator–classifier pair. Positive values indicate that a metric score adds information; negative values indicate net noise.	38

References

- [1] H. Zhang, H. Song, S. Li, M. Zhou, and D. Song, “A survey of controllable text generation using transformer-based pre-trained language models,” *ACM Comput. Surv.*, vol. 56, Oct. 2023.
- [2] M. Ku, T. Li, K. Zhang, Y. Lu, X. Fu, W. Zhuang, and W. Chen, “Imagenhub: Standardizing the evaluation of conditional image generation models,” in *ICLR*, 2024.
- [3] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu, “Plug and play language models: A simple approach to controlled text generation,” in *International Conference on Learning Representations*, 2020.
- [4] B. Krause, A. D. Gotmare, B. McCann, N. S. Keskar, S. Joty, R. Socher, and N. F. Rajani, “GeDi: Generative discriminator guided sequence generation,” in *Findings of the Association for Computational Linguistics: EMNLP 2021* (M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, eds.), (Punta Cana, Dominican Republic), pp. 4929–4952, Association for Computational Linguistics, Nov. 2021.
- [5] M. Khalifa, H. Elshahar, and M. Dymetman, “A distributional approach to controlled text generation,” in *International Conference on Learning Representations*, 2021.
- [6] H. Zhang and D. Song, “DisCup: Discriminator cooperative unlikelihood prompt-tuning for controllable text generation,” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing* (Y. Goldberg, Z. Kozareva, and Y. Zhang, eds.), (Abu Dhabi, United Arab Emirates), pp. 3392–3406, Association for Computational Linguistics, Dec. 2022.

- [7] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, “Hierarchical text-conditional image generation with clip latents,” *arXiv preprint arXiv:2204.06125*, vol. 1, no. 2, p. 3, 2022.
- [8] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.
- [9] X. Liang, H. Wang, Y. Wang, S. Song, J. Yang, S. Niu, J. Hu, D. Liu, S. Yao, F. Xiong, and Z. Li, “Controllable text generation for large language models: A survey,” 2024.
- [10] S. Hartwig, D. Engel, L. Sick, H. Kniesel, T. Payer, P. Poonam, M. Glöckler, A. Bäuerle, and T. Ropinski, “A survey on quality metrics for text-to-image generation,” 2025.
- [11] G. Forman, “Quantifying counts and costs via classification,” *Data Mining and Knowledge Discovery*, vol. 17, pp. 164–206, 2008.
- [12] P. von Däniken, J. M. Deriu, A. Rodrigo, and M. Cieliebak, “Improving quantification with minimal in-domain annotations: Beyond classify and count,” *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 18, pp. 1585–1598, May 2024.
- [13] A. Bella, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana, “Quantification via probability estimators,” in *2010 IEEE International Conference on Data Mining*, pp. 737–742, 2010.
- [14] S. Wu and P. Resnick, “Calibrate-extrapolate: Rethinking prevalence estimation with black box classifiers,” *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 18, pp. 1634–1647, May 2024.
- [15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [16] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-end object detection with transformers,” in *Computer Vision – ECCV 2020* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), (Cham), pp. 213–229, Springer International Publishing, 2020.
- [17] Z. Lin, D. Pathak, B. Li, J. Li, X. Xia, G. Neubig, P. Zhang, and D. Ramanan, “Evaluating text-to-visual generation with image-to-text generation,” *arXiv preprint arXiv:2404.01291*, 2024.
- [18] Y. Hu, B. Liu, J. Kasai, Y. Wang, M. Ostendorf, R. Krishna, and N. A. Smith, “Tifa: Accurate and interpretable text-to-image faithfulness evaluation with question answering,” *arXiv preprint arXiv:2303.11897*, 2023.
- [19] Y. Chen, B. Xu, Q. Wang, Y. Liu, and Z. Mao, “Benchmarking large language models on controllable generation under diversified instructions,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 17808–17816, Mar. 2024.
- [20] D. Ashok and B. Póczos, “Controllable text generation in the instruction-tuning era,” 2024.
- [21] S. Gehman, S. Gururangan, M. Sap, Y. Choi, and N. A. Smith, “RealToxicityPrompts: Evaluating neural toxic degeneration in language models,” in *Findings of the Association for Computational Linguistics: EMNLP 2020* (T. Cohn, Y. He, and Y. Liu, eds.), (Online), pp. 3356–3369, Association for Computational Linguistics, Nov. 2020.

- [22] P. Ke, H. Zhou, Y. Lin, P. Li, J. Zhou, X. Zhu, and M. Huang, “CTRLEval: An unsupervised reference-free metric for evaluating controlled text generation,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (S. Muresan, P. Nakov, and A. Villavicencio, eds.), (Dublin, Ireland), pp. 2306–2319, Association for Computational Linguistics, May 2022.
- [23] L. Binyamin, Y. Tewel, H. Segev, E. Hirsch, R. Rassin, and G. Chechik, “Make it count: Text-to-image generation with an accurate number of objects,” *arXiv preprint arXiv:2406.10210*, 2024.
- [24] C.-Y. Wang and H.-Y. M. Liao, “YOLOv9: Learning what you want to learn using programmable gradient information,” 2024.
- [25] S. Rosenthal, N. Farra, and P. Nakov, “SemEval-2017 task 4: Sentiment analysis in Twitter,” in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)* (S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. Cer, and D. Jurgens, eds.), (Vancouver, Canada), pp. 502–518, Association for Computational Linguistics, Aug. 2017.
- [26] P. Nakov, A. Ritter, S. Rosenthal, F. Sebastiani, and V. Stoyanov, “SemEval-2016 task 4: Sentiment analysis in Twitter,” in *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)* (S. Bethard, M. Carpuat, D. Cer, D. Jurgens, P. Nakov, and T. Zesch, eds.), (San Diego, California), pp. 1–18, Association for Computational Linguistics, June 2016.
- [27] J. Deriu, P. von Däniken, D. Tugener, and M. Cieliebak, “Correction of errors in preference ratings from automated metrics for text generation,” in *Findings of the Association for Computational Linguistics: ACL 2023* (A. Rogers, J. Boyd-Graber, and N. Okazaki, eds.), (Toronto, Canada), pp. 6456–6474, Association for Computational Linguistics, July 2023.
- [28] P. von Däniken, J. Deriu, D. Tugener, and M. Cieliebak, “On the effectiveness of automated metrics for text generation systems,” in *Findings of the Association for Computational Linguistics: EMNLP 2022* (Y. Goldberg, Z. Kozareva, and Y. Zhang, eds.), (Abu Dhabi, United Arab Emirates), pp. 1503–1522, Association for Computational Linguistics, Dec. 2022.
- [29] E. GmbH, “Prodigy an annotation tool for ai, machine learning nlp,” 2025.
- [30] J. Cohen, “A coefficient of agreement for nominal scales,” *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960.
- [31] J. R. Landis and G. G. Koch, “The measurement of observer agreement for categorical data,” *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.
- [32] N. Mostafazadeh, N. Chambers, X. He, D. Parikh, D. Batra, L. Vanderwende, P. Kohli, and J. Allen, “A corpus and evaluation framework for deeper understanding of commonsense stories,” *arXiv preprint arXiv:1604.01696*, 2016.
- [33] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Wyatt, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Guzmán, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Thattai, G. Nail, G. Mialon, G. Pang, G. Curell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra,

- I. Evtimov, J. Zhang, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla, K. Lakhotia, L. Rantala-Yearly, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsimpoukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen, S. Wan, S. Bhosale, S. Zhang, S. Vandenhende, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou, T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. Wang, X. E. Tan, X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Srivastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Baevski, A. Feinstein, A. Kallet, A. Sangani, A. Teo, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardi, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Hancock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia, C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichtenhofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel, F. Caggioni, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Inan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, H. Zhan, I. Damla, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U, K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani, M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. Liu, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy, R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Patil, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max,

- S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wu, X. Wang, X. Wu, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, and Z. Ma, “The llama 3 herd of models,” 2024.
- [34] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, “Llama 2: Open foundation and fine-tuned chat models,” 2023.
- [35] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, “Mistral 7b,” 2023.
- [36] A. Hazourli, “Financialbert - a pretrained language model for financial text mining,” 02 2022.
- [37] P. Malo, A. Sinha, P. Korhonen, J. Wallenius, and P. Takala, “Good debt or bad debt: Detecting semantic orientations in economic texts,” *J. Assoc. Inf. Sci. Technol.*, vol. 65, p. 782–796, Apr. 2014.
- [38] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, D. Podell, T. Dockhorn, Z. English, and R. Rombach, “Scaling rectified flow transformers for high-resolution image synthesis,” in *Proceedings of the 41st International Conference on Machine Learning*, ICML’24, JMLR.org, 2024.
- [39] P. Pernias, D. Rampas, M. L. Richter, C. Pal, and M. Aubreville, “Würstchen: An efficient architecture for large-scale text-to-image diffusion models,” in *The Twelfth International Conference on Learning Representations*, 2024.
- [40] Black Forest Labs, “Flux.” <https://github.com/black-forest-labs/flux>, 2024.
- [41] H. Liu, C. Li, Q. Wu, and Y. J. Lee, “Visual instruction tuning,” in *Advances in Neural Information Processing Systems* (A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds.), vol. 36, pp. 34892–34916, Curran Associates, Inc., 2023.