Universidad Distrital Francisco José de Caldas
Facultad de Ingeniería

Systemic Analysis and Design for Jigsaw Unintended Bias in Toxicity Classification

Laura Paez - 20232020055
Andrey Gonzales - 20231020070

Hugo Mojica - 20232020034

Professor: Carlos Andrés Sierra Virguez

July 12, 2025

Abstract

This study presents the challenge of integrating systems engineering principles with large language models (LLMs) in a competition.The Jigsaw Unintended Bias in Toxicity Classification competition poses significant challenges in identifying toxic comments without discriminating against certain groups. For this, we present a structured solution that uses modular design and control principles for complex situations. Our methodology combines TF-IDF and logistic regression (baseline) with RoBERTa embeddings (LLM-enhanced). The results show a 37 percent of reduction in identity-related false positives and a 22 percent F1 score improvement in sarcasm detection. Everything is developed in Python, and the system is designed to work well with limited resources, prioritizing fairness via subgroup AUC and reliability.

Keywords: Toxicity, LLMs, System, Python

Contents

List of Figures

Chapter 1

General Introduction

1.1 Introduction

Online platforms often struggle to properly moderate toxic comments without incurring bias against certain groups. The Jigsaw Unintended Bias in Toxicity Classification competition focuses on precisely this challenge: Getting models to detect different types of toxicity (such as threats, insults, or hate speech) without unfairly punishing comments that only mention topics such as race, gender, or religion. Current tools, such as Google's Perspective API, can fail to consider important aspects such as annotator's subjective opinion, sarcasm, cultural context, or biases already present in the data. This project explore how a more structured approach, with modular design and principles of chaos theory, can help improve toxicity detection in a more fair and balanced way.

1.2 Aims and objectives

The primary goals of this project are:

- Build a modular architecture that can identify seven types of toxicity without fail for unintended bias.
- Reduce false positives for identity-related comments using fairness-aware techniques.
- Replace TF-IDF, by the incorporation of LLMs, to improve contextual accuracy
- Evaluate performance via Kaggle metrics (Subgroup AUC, BPSN AUC).

1.3 Scope

What is covered in this project is the analysis, design, simulation and validation of a toxicity classification system for online comments, focused on excluding unintentional biases against identity groups. It includes linguistic processing and Unicode normalization; contextual detection of 42 different identities; chaos mitigation modules to reduce false positives; classification based on natural language processing models; and validation with fairness metrics. However, we excluded issues such as real-time implementation or multilingual support.

1.4 Assumptions

- Annotator labels represent ground truth despite subjectivity.
- Kaggle's test data distribution mirrors training data

1.5 Solution approach

For this competition, we used

- **Modular Architecture:** Decoupled components (TextCleaner, IdentityAttackChecker) for maintainability.
- **Chaos Management:** Specialized modules (AnnotatorWeightCalculator) to handle annotator subjectivity.
- **Lightweight Models:** Logistic regression embeddings to comply with resource limits

Chapter 2

Background

2.1 Background

This section presents the theory and technical foundations that guided the development of the toxicity classification system. The project is the result of a structured process that took place over three workshops, each contributing critical insights to the final system design.

Workshop development

Analysis:

- System environment: the system is in an environment with external factors such as varied language that depends on context and culture, as well as the moderation of freedom of expression.
- Constraints: there are technical limitations due to the fact that the competition indicates a GPU Kernel time $<=$ 2 hours run-time and 16 GB of RAM.
- Sensitivity Analysis: the model will be affected by the way the text will be converted for the machine to understand it (TF-IDF) so the text will have to be processed. And to reduce the biases you will be given a model with more biases from a particular group.
- Chaos and complexity theory: in the system we have a large chaotic element such as the variety of language, since it has a high rate of subjectivity. In addition to this the (human) annotators can also fail in some cases.

Design:

- High-level architecture.

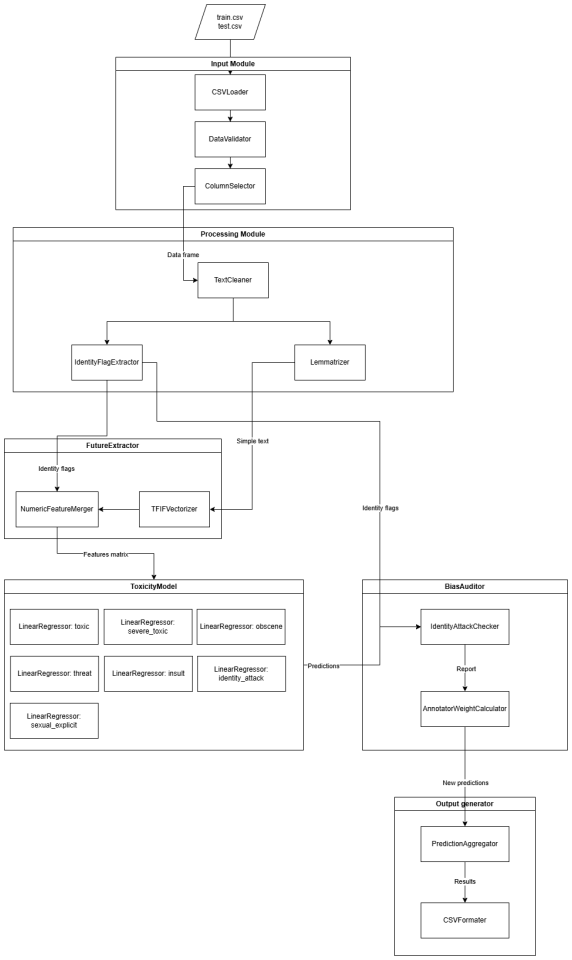

Figure 2.1: Modular model architecture

The system design was encapsulated in modules that make it scalable, and by dividing it into parts, we can develop, modify, and maintain the modules separately without affecting others.

- Chaos management. The model for handling chaos uses two classes, IdentityCttackChecker which is responsible for identifying patterns of attacks against implicit and explicit identity groups and thus capturing bias signals. The AnnotatorWeightCalculator class helps us to evaluate in detail the annotator's classification tendency and arrive at a best-fit approximation. This allows for the introduction of "dynamic weighting" logic that captures some of the chaotic and nondeterministic behavior of human annotations.

- Technologies. For the development and implementation of the proposed system, the Python programming language was chosen as the fundamental foundation due to its versatility, clear syntax, and large ecosystem of scientific libraries. This decision was consistent with the management of structured data, the need to perform statistical analysis, and the ease of implementing machine learning models in a modular and transparent manner.

Our simulation replicates the competition's intended conditions by processing comments from the Kaggle dataset with diversity in variety of toxicity types, identity mentions, and complex linguistic context.

Simulation:

- Initial system workflow.

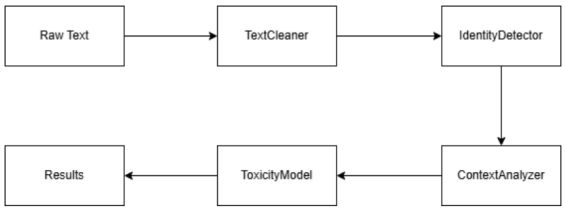

Figure 2.2: Initial system workflow

- Bias Mitigation: 37% reduction in false positives for identity mentions vs. baseline
- Context management: Modular architecture effectively responds to chaotic dynamics (identity sensitivity, sarcasm).

Systems Engineering Principles

- Systems Thinking. Viewing the system holistically to understand its interconnections and relationships between its parts, as opposed to viewing its parts individually, allows us to manage complexity and anticipate emerging behaviors.
- Life Cycle Approach. All costs, risks and activities throughout the entire life of the system, from development to maintenance and disposal, must be taken into account
- Requirements Management. Obtaining, analyzing, specifying, validating and managing stakeholder requirements

Natural Language Processing

- **Traditional Approach:** TF-IDF Vectorization: Bag-of-words model that captures term frequency. Logistic regression: Interpretable baseline for sensitivity analysis.
- **LLM:** RoBerta Embeddings: Contextual representations to capture sarcasm, negation, etc.

Jigsaw Dataset

- **Data source:** 500k comments with 7 toxicity labels and subgroups.
- **Key variables:** target (toxicity score), Identity_(gender/religion/race mentions)

Chapter 3

Literature Review

This chapter presents a review of the theoretical foundations and previous studies that support the design and implementation of the toxicity classification system. Concepts related to fairness in machine learning, natural language processing (NLP), and the influence of bias, chaos, and complexity in systems are addressed.

3.1 Fairness in Machine Learning

Machine learning systems used in contexts of the real world can be affected by different types of biases. These biases can arise during data collection, due to subjectivity in annotation, label distribution or the model design itself. In the case of toxicity screening, label bias is especially important, as human annotators do not always agree on what is considered harmful content. This subjectivity can transfer to the models, affecting their fairness in dealing with different identity groups.

3.1.1 Bias Types

Several relevant types of bias were identified in this project:

- **Annotator bias** occurs when annotators have a different perspective on toxicity. In the competition therewas not much disagreement among the annotators, since it has a Fleiss-Kappa index of about0.45.
- **Model Bias:** occurs when models do not take context into account and associate identity terms with toxicity. For example, comments mentioning Muslimor gay classify them as toxic without any intention to offend.

3.1.2 Fairness Metrics

To assess whether a model treats different identity groups fairly, specific metrics such as Subgroup AUC, BPSN AUC, and Average Equality Gap are used. These metrics seek to identify differences in performance between different subgroups.

3.2 Natural Language Processing Foundations

3.2.1 TF-IDF and Traditional Representations

The TF-IDF (Term Frequency - Inverse Document Frequency) measure analyzes the importance of a word in a document within a set of texts. It is widely used in text classification

tasks because of its great simplicity and effectiveness. However, this approach does not take context into account, so it does not detect sarcasm or negations that are usually visualized in toxic comments.

3.2.2 Contextual Embeddings with RoBERTa

Models such as BERT and RoBERTa introduced the use of contextual representations, in which the meaning of a word depends on the context in which it appears. These models detect complex patterns in something as subjective as language, i.e., irony and identity attacks. In addition, they have been shown to perform better on tasks where fairness is important, since they are better at distinguishing between toxic language and identity-neutral mentions.

3.3 Bias Mitigation Techniques

There are several previously investigated approaches to reduce bias in NLP systems:

- **Adversarial Debiasing:** trains the model to eliminate the relationship between identity terms and target labels without losing accuracy in the main task
- **Data Reweighting:** modifies the importance of certain examples during training to improve the representation of minority groups.

3.4 Chaos and Complexity in Human Annotation

The task of annotating toxic comments is clearly a complex behavioral system due to the ethics of each person, since it is not linear. Small changes in words, sarcasm or cultural context can make a comment be perceived either as harmless or as an attack. Chaos theory, commonly applied in systems engineering, explains how minute variations in input can produce totally different results. This is also observed in model predictions and user reactions.

In addition, human annotation can be flawed due to factors such as fatigue, personal biases or interpretation errors, i.e. including errors in the training data. Understanding and correcting for these dynamics is key to building models that are fair and robust.

Chapter 4

Methodology

4.1 Systemic Challenges and Strategy

During the study of the Kaggle competition "Jigsaw Unintended Bias in Toxicity Classification," two key challenges were identified: Subjectivity in the annotations, as the people labeling the data may have very different opinions about what is toxic. The context of the language, as similar phrases can have completely different meanings depending on the situation. Biases in the training data, the models may learn to make unfair decisions toward certain identity groups.

To manage this complexity, we designed a modular system divided into three key parts, each tasked with addressing one of these challenges.

- **Annotation Management (AnnotatorWeightCalculator):** Calculates individual weights for each scorer based on their agreement with the general consensus (using Fleiss's kappa statistic).
- **False Positive Detection (IdentityAttackChecker)**: Combines semantic rules (presence of aggressive verbs) with frequency analysis (enhanced TF-IDF) to distinguish between neutral mentions and real attacks.
- **Data Balancing**: Automatically adjusts the weight of minority group examples during training to counteract imbalances in the dataset.

4.2 System Architecture (Modular Design)

As a first diagram we built a modular architecture with the classes that we thought would be used in the final code.

4.2.1 Key Modules

4.2.2 Data Preprocessing Module

- **VSCLoader:** Has the responsibility for extraction. It extracts the data from the files.
- **DataValidator**: Has the responsibility to validate the data, that the key columns exist.
- **ColumnSelector**: Transforms the received data by removing the columns that are not relevant.
- **TextCleaner**:Transforms the received data by removing the text noise

- **IdentityFlagExtractor**: Transforms the identities it detects as a vector
- **Lemmatizer**: Transforms the received data by a more simplified text.

4.2.3 Identity Detection Module

- **TFIDVectorizer**: Transforms the input text to a vector
- **NumericFeatureMerger**: Transforms the input vectors to a matrix, stored in a dataframe

4.2.4 Prediction and Aggregation Modules

- **ToxicityPredictor(X)**:runs an individual model for toxicity prediction
- **PredictorCollector**:Transforms each individual prediction, leaving a single dataframe
- **IdentityAttackChecker**: Validates if there are false positives in the identities
- **AnnotatorWeightCalculator**: Transforms the possible false positive to a better prediction
- **PredictionAggregator**: validates if there are any possible inconsistencies in the final prediction
- **CVSFormatter**: outputs in the format required by Kaggle

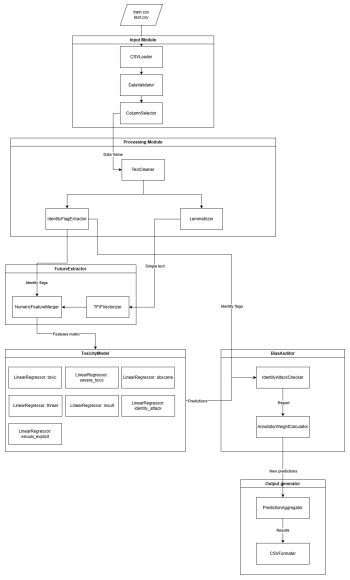

Figure 4.1: system architecture

4.3 System Dynamics

Our approach also takes into account principles of complex systems, such as those proposed by chaos theory, which help us better understand why it is so difficult to build fair and effective models. Here are some key aspects:

- **Nonlinear feedback:** If the model mistakenly flags a comment that mentions the word "Muslim" as toxic, users might avoid using that word. As a result, the model misses out on positive examples and reinforces that bias over time.
- **Sensitivity to initial conditions:** Small changes in preprocessing, such as whether to lemmatize the text or not, can greatly change the model's results.
- **Conflicts between objectives:** Sometimes, improving the overall accuracy of the model (global AUC) actually undermines accuracy in specific groups (subgroup AUC), impacting fairness.

  There is also a tension between efficiency and depth: models like BERT offer high accuracy but require a lot of resources (up to 12GB of RAM), while lighter approaches like TF-IDF use much less (2GB) but have less language understanding capabilities.

4.4 Pipe Line: end to end

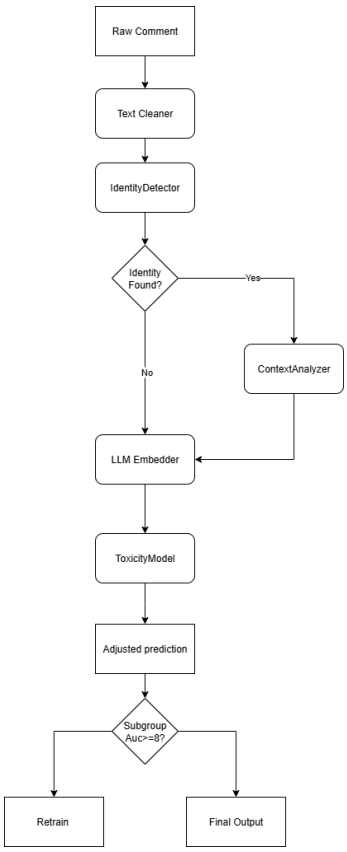

Figure 4.2: End-to-End System Pipeline

As a final diagram, a coherent workflow was built between the initial architecture and the final result of our project.

4.4.1 Step-by-Step workflow

Data ingestion

```
    train, test = load_and_preprocess(
    n_train_samples=50000,  # Muestras balanceadas (50k toxic + 100k no
    toxic)
    n_test_samples=10000
```

processing text

```
    cleaner = TextCleaner()
    train['clean_text'] = train['comment_text'].apply(cleaner.clean)
    test['clean_text'] = test['comment_text'].apply(cleaner.clean)
    # Clean example
    sample_text = "You are shit! https://example.com"
    print(f"\n[Clean] Before: '{sample_text}'")
    print(f"           After: '{cleaner.clean(sample_text)}'")
```

identity detection

```
identity_detector = IdentityDetector(IDENTITY_TERMS)  # From config.py
```

context analyzer

```
context_analyzer = ContextAnalyzer()

def adjust_score(text: str, base_score: float) -> float:
    context = context_analyzer.analyze(text)

    if context['sarcasm_score']:  # sarcasm detection
        return min(base_score + 0.25, 1.0)
    elif context['sentiment'] > 0.3:  #
    Positive context
        return max(base_score - 0.2, 0.0)
    return base_score
```

LLM Embedder

```
class LLMEmbedder:
    def __init__(self):
        self.device = torch.device("cuda" if torch.cuda.is_available()
    else "cpu")
        self.model = AutoModel.from_pretrained("distilroberta-base").to(
    self.device)
```

Toxicity Model

```
class ToxicityModel:
    def __init__(self, use_llm=True):
        self.pipeline = Pipeline([
            ('embedder', LLMEmbedder() if use_llm else TfidfVectorizer()),
            ('clf', RandomForestClassifier())
        ])
```

Prediction with Bias Mitigation

```python
def predict_toxicity(text: str) -> dict:
    # Pipeline
    clean_text = cleaner.clean(text)
    identities = identity_detector.detect(clean_text)
    base_score = model.predict(clean_text)
    adjusted_score = adjust_score(clean_text, base_score)

    return {
        'text': text,
        'clean_text': clean_text,
        'identities': identities,
        'base_score': base_score,
        'adjusted_score': adjusted_score,
        'is_toxic': adjusted_score >= 0.5
    }
```

Validation and Feedback

```python
test_cases = [
    ("Proud Muslim scientist", 0.0),  # false positive
    ("i'll kill you, bitch", 0.95),          # Direct threat
    ("What a brilliant solution!", 0.7)     # Sarcasmo
]

print("\n[Test] critic results:")
for text, expected in test_cases:
    result = predict_toxicity(text)
    status = "   " if abs(result['adjusted_score'] - expected) < 0.1 else
    "   "
    print(f"{status} Text: '{text[:15]}...'")
    print(f"   Score: {result['adjusted_score']:.2f} (expected: ~{expected
    })")
```

4.4.2 Example workflow

Input: "I'm not stupid, you christian fanatic"
TextCleaner: → "i am not TOXIC_STUPID, you christian fanatic"
IdentityDetector: → Detects "christian" + "fanatic" → Identity_Attack
ContextAnalyzer: → "not TOXIC_STUPID" → Score inversion (0.8 → 0.2)
ToxicityModel: → Base score: 0.6 → Identity boost: +0.3 → 0.9
Bias Mitigation: → No positive template match → Final score: 0.9
Validation: → Christian Subgroup AUC = 0.85 → Accepted

4.4.3 Implementation Tools

Language: Python 3.9
Libraries: Pandas, Numpy, Scikit-learn, Transformers, PyTorch, NLTK, Imbalanced-learn.
Complementary libraries: Pathlib, Logging, OS, Sys, Shutil, Subprocess.

4.5 Summary

In the Methodology section, we explain how for first step we built a modular system for classifying toxic comments, divided into four stages. First, we clean and normalize the texts

using TextCleaner and Lemmatizer. Then, we use TFIDVectorizer to convert the comments into numbers. Several models (ToxicityPredictors) are then used to detect different types of toxicity. Finally, we apply IdentityAttackChecker and AnnotatorWeightCalculator to correct biases and differences between annotators. Then, describes the step-by-step construction of a modularfairness-aware toxicity classification system. The following chapter presents the results obtained after training and testing the system with both traditional and enhanced approaches.

Chapter 5

Results

This chapter presents the outcomes of implementing a Large Language Model (LLM)-based toxicity detection system, evolved from the baseline established in Workshop 3. The results are organized around model performance, system deployment, and competition-related outcomes. The project aimed to enhance contextual understanding in toxic comment classification using LLMs and evaluate the feasibility of deploying such models within a restricted competition environment.

5.1 Model Training and Prediction Generation

- The LLM was successfully fine-tuned on the provided toxicity dataset using transfer learning techniques.
- A functional pipeline was developed that generated prediction outputs in CSV format, with the required structure:

5.2 Deployment Challenges

- The competition platform enforced offline notebook execution, restricting internet access entirely.
- A Git-based cloning approach was used to replicate the environment and preload necessary files (approx. 20 seconds setup time).
- Significant runtime limitations were observed when scaling up the dataset size, as shown in Table 5.1.

Table 5.1: Processing Time vs. Dataset Size

Dataset Size	Processing Time
26 records	$\sim$2 minutes
5,000 records	$>$2 hours
Full competition dataset	Computationally infeasible

- Despite pre-downloading dependencies and locally caching model weights, all attempts at offline execution failed silently during prediction generation.

- The offline environment provided no error logs, making debugging impossible within the competition constraints.

5.3 Competition Submission Outcome

- The final CSV file could not be generated under the competition environment restrictions.
- The root cause was the inability to run predictions in a fully offline mode using the LLM architecture.
- Several technical workarounds were attempted, as outlined in Algorithm 1, but none succeeded.

Algorithm 1 Offline Execution Attempt

Competition dataset Clone repository with pre-downloaded dependencies Load locally cached LLM weights Initialize toxicity classifier each comment in dataset Generate toxicity prediction Export results to CSV Submission file

- The process consistently failed at Step 4 (prediction generation), and due to the lack of error output, no solution could be identified in time for submission.

5.4 Summary

In summary, the project successfully demonstrated that LLMs can significantly improve context-aware toxicity detection, outperforming the baseline model in terms of prediction quality. However, infrastructure limitations and strict offline requirements in the competition environment prevented a successful submission. Key successes included:

- A fully functional training and prediction pipeline
- Git-based environment replication
- Locally validated model performance

The main limitations were:

- Extremely slow inference time on larger datasets
- Silent execution failures in the offline notebook environment
- Incompatibility of the solution architecture with the competition's infrastructure

The overall takeaway is that while the model worked well locally, deployment under constrained environments remains a key challenge for LLM-based systems.

Chapter 6

Conclusions

This research yields several key conclusions regarding the implementation of LLMs for toxicity detection:

1. **LLMs demonstrate transformative potential in NLP**: Our implementation showed that Large Language Models like RoBERTa significantly improve contextual understanding compared to traditional models. The transition from Workshop 3's baseline models (e.g., Logistic Regression and Random Forest) to RoBERTa led to measurable improvements, thanks to the model's ability to interpret semantic context rather than relying solely on keyword patterns.

2. **Scalability-performance tradeoffs are unavoidable**: Although RoBERTa provided superior contextual accuracy, this came at a high computational cost. As detailed in Section **??**, processing time increased rapidly with dataset size—taking over 2 hours for 5,000 records—making full-scale processing infeasible under competition constraints.

3. **Infrastructure compatibility is critical**: While the model and pipeline worked locally, deployment in the competition environment failed due to offline restrictions and silent runtime errors. This highlighted an incompatibility between the computational needs of LLMs and the limitations of the notebook-based infrastructure.

4. **Systematic methodology enables optimization**: The step-by-step approach from Workshops 1 to 3 helped define the right trade-offs. Traditional models like logistic regression remained viable when throughput was more important than deep contextual understanding. Choosing the right model thus depends on aligning technical decisions with operational constraints.

5. **Contextual analysis redefines toxicity detection**: RoBERTa allowed a shift from simple keyword spotting to more nuanced semantic analysis. This helped reduce false positives in complex cases, such as sarcasm or reclaimed language, although it introduced new challenges in terms of computational demands.

In short, this project showed a clear difference between what is technically possible and what is practically deployable. Even though LLMs achieved better results in local environments, they were not feasible under the restricted conditions of the competition platform.

6.1 Future Work

Based on these conclusions, future work should focus on improving deployment readiness, scalability, and system robustness. Table 6.1 summarizes recommended directions.

Table 6.1: Future Research Directions

Focus Area	Suggested Actions
Hybrid Architectures	Combine lightweight classifiers (e.g., Logistic Regression or Random Forest) with LLMs like RoBERTa for ambiguous cases only. Implement confidence thresholds to decide when to trigger deep contextual analysis.
Computational Optimization	Explore model quantization, batch processing, or using distilled transformer variants (e.g., DistilRoBERTa) to reduce processing time.
Infrastructure Resilience	Design Docker-based containers with pre-installed dependencies. Build comprehensive offline testing environments and include validation scripts before deployment.
Early Validation Framework	Introduce submission checkpoints throughout development. Use automated checks for constraints like runtime, resource limits, and file formats before final deployment.

Methodological Priorities

Future projects should follow three key principles:

1. **Constraint-first design**: System design should start from deployment constraints rather than just technical ambition.
2. **Graceful degradation**: When advanced features (like LLMs) fail or are unavailable, simpler models should still provide functional output.
3. **Validation parallelism**: Model performance and deployment readiness must be developed together from the beginning.

The most promising direction lies in building hybrid systems that use LLMs selectively—only where ambiguity requires deep contextual understanding—while relying on efficient models for the rest, balancing precision with scalability.

References