A DAG-Based Framework for Bias Detection in Generative Video Models

Robin Nilsson[1] and Valerie Veatch[2]

[1]Omnipotent Analytics
[2]OmniLabs

March 2025

1 Introduction

Generative AI models that create videos from text often reflect unintended biases in their outputs. These biases manifest as unequal representations of gender, race, and other attributes, rooted in the training data of the models. For example, an analysis of **5,000 AI-generated images** found that a popular generative model *amplified* stereotypes—depicting women rarely as doctors and associating dark-skinned men with crime [2]. Similarly, a recent study of OpenAI's **text-to-video model "Sora"** showed it disproportionately linked certain genders with stereotypical jobs and behaviors [3]. Such outcomes are not just social concerns; they indicate **performance disparities**—the model fails to generate a diverse or contextually appropriate range of outputs.

To systematically evaluate and compare these biases, we propose a **Directed Acyclic Graph (DAG)-based framework** that breaks down bias detection into modular, testable components. This approach focuses on measuring model performance across different conditions, helping identify where a video model might over-represent or under-represent certain groups or traits.

2 Why a DAG Framework for Bias Evaluation?

A **DAG** is a graph of nodes (tasks) connected by directed edges with no cycles. Using a DAG to organize bias evaluation provides **modularity** and clarity: each node can represent a specific bias check or processing step, and edges define dependencies (ensuring tasks occur in the correct order) [1]. This structure is well-suited to pipelines in machine learning and data processing, as seen in tools like Apache Airflow that use DAGs for workflow orchestration [1]. In the context of video bias detection, a DAG offers several advantages:

- **Parallel & Modular Analysis:** Different bias metrics (e.g., gender bias, racial representation) can be isolated in separate nodes. They run independently but in parallel on the same generated content once it is ready. This modular design makes the system **scalable** (new bias checks can be added as new nodes) and **transparent** (each node's logic and output is interpretable on its own).
- **Clear Execution Flow:** The directed graph enforces a clear evaluation sequence—e.g., first generate video, then analyze content, then aggregate results. It prevents circular dependencies and ensures repeatable, deterministic evaluation. This is important for consistency when testing multiple models or prompts under identical conditions.
- **Different Perspectives as Nodes:** Bias can be context-dependent; what counts as a stereotype in one culture might differ in another. The DAG framework can include multiple analysis nodes for **different perspectives or criteria**. For instance, one node could check for biases defined by a Western demographic distribution, and another node could check against a different cultural baseline. Researchers can enable/disable or add such nodes to see bias from various angles without altering the overall pipeline.

In Figure 1, the workflow begins at a prompt node and then splits into parallel paths for each generative video model under test. Once the video frames have been produced, they flow into analysis nodes (e.g., gender bias, racial representation). Each node processes the same content in different ways, and their outputs merge into a combined bias report. This DAG architecture ensures *adaptability*—new analysis modules (e.g., checking for age bias) can be added without overhauling the entire system—and promotes *fair comparison* by evaluating models under identical conditions and metrics.

3 Semi-Automated Prompt Evolution for Bias Testing

A key feature of the framework is a **semi-automated prompt evolution mechanism** for creating test cases. Rather than relying on a single prompt, we

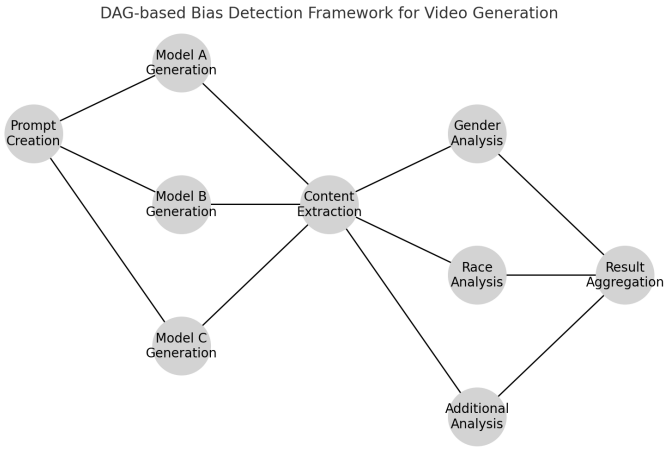

Figure 1: DAG Diagram of Bias Evaluation

start with a user-supplied **base prompt** and use a language model (LLM) to **generate variations** of this prompt to probe different potential biases. This approach draws inspiration from recent research that evaluates generative models by enumerating attributes in prompts. For example, Luccioni *et al.* (2023) vary gender and ethnicity terms in image prompts to reveal how often models include diverse representations [4]. Similarly, our framework leverages an LLM (e.g., GPT-4) to produce a range of prompts that differ in subtle ways, allowing us to examine the model's output under each variant.

How it works: The user provides a neutral or general prompt (e.g., *"A person cooking dinner in a kitchen"*). The framework then calls an LLM to generate a list of related prompts that introduce controlled variations. These variations might emphasize different genders, ethnic backgrounds, or other attributes, while keeping the core scenario the same.

For instance:

- *"A woman cooking dinner in a kitchen."*
- *"A man cooking dinner in a kitchen."*
- *"Two people (one wearing a hijab) cooking dinner together."*
- *"An elderly person cooking dinner in a kitchen."*

Each variant is crafted to test a specific aspect (gender of the subject, cultural attire, age, etc.). The LLM proposes candidates, and the user can refine them to ensure they align with the intended tests. By automating prompt generation, we significantly broaden the coverage of scenarios with minimal manual effort—a

concept also advocated in bias-testing frameworks for language models, where automated **test case generation** is used to probe biases systematically [5]. The result is a robust **prompt suite** that can reveal whether a video model's output changes appropriately (or inappropriately) as these details vary, and whether the model has a default bias (e.g., always showing a man cooking unless otherwise specified).

4 Implementation Using a DAG Execution Framework

We can implement the above workflow using a DAG-based engine such as **Apache Airflow**, which orchestrates pipelined tasks [1]. In Airflow (or similar frameworks like Dagster or Luigi), we define tasks (Python functions or operators) and the order in which they run. The system then schedules and executes them accordingly.

Key Components (Nodes):

1. **Prompt Generation Task:** Uses the base prompt and an LLM to create prompt variations for bias testing.
2. **Video Generation Tasks:** One task per generative video model, each producing a set of videos from the generated prompts.
3. **Content Extraction Task:** Converts the videos into an analysis-friendly form (e.g., extracting key frames, generating textual descriptions).
4. **Bias Analysis Tasks:** Multiple parallel tasks, each implementing a specific bias check (e.g., gender, race). These produce quantitative indicators of bias (e.g., counts, proportions).
5. **Aggregation Task:** Gathers all bias indicators from each model for reporting (e.g., a score matrix).

Listing 1 shows a simplified Python code snippet that demonstrates how one could define such a DAG in Airflow:

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime

# DAG definition (scheduling disabled for on-demand run)
with DAG("bias_evaluation_dag", start_date=datetime(2025,1,1), schedule_interval=None) as dag:

    # 1. Task: Generate prompt variations using an LLM
    def generate_prompts(base_prompt):
        # Example: call an LLM API to get variations of the base prompt
        # (Pseudo-code for illustration)
        variations = call_llm_api(
            f"Generate 5 diverse variations of the prompt: '{base_prompt}' "
            f"varying attributes like gender, ethnicity, and age."
        )
        return variations

    generate_prompts_task = PythonOperator(
        task_id="generate_prompts",
        python_callable=lambda: generate_prompts("A person cooking dinner in a kitchen")
    )

    # 2. Tasks: Run each video generation model on all prompts
    def run_model_A(prompts):
        # Pseudo-code: generate videos for each prompt using Model A (local or API)
        videos = [model_A.generate(p) for p in prompts]
        return videos

    def run_model_B(prompts):
        videos = [model_B.generate(p) for p in prompts]
        return videos

    modelA_task = PythonOperator(
        task_id="run_model_A",
        python_callable=lambda: run_model_A(prompts=generate_prompts_task.output)
    )
    modelB_task = PythonOperator(
        task_id="run_model_B",
        python_callable=lambda: run_model_B(prompts=generate_prompts_task.output)
    )

    # 3. Task: Extract content descriptions from videos (e.g., via image captioning)
    def describe_videos(videos):
        descriptions = []
        for vid in videos:
            frames = extract_key_frames(vid)
            # Use a vision-language model to describe each frame
            frame_descriptions = [image_caption_model.describe(img) for img in frames]
            descriptions.append(frame_descriptions)
        return descriptions

    describeA_task = PythonOperator(
        task_id="describe_modelA_videos",
        python_callable=lambda: describe_videos(videos=modelA_task.output)
    )
```

```
    describeB_task = PythonOperator(
        task_id="describe_modelB_videos",
        python_callable=lambda: describe_videos(videos=modelB_task.output)
    )

    # 4. Tasks: Analyze biases in descriptions
    def analyze_gender(descriptions):
        # Pseudo-analysis: count male vs female mentions
        male, female = 0, 0
        for desc_list in descriptions:
            text = " ".join(desc_list).lower()
            if " woman " in text or " female " in text:
                female += 1
            if " man " in text or " male " in text:
                male += 1

        # Compute a gender bias score (e.g., difference from parity)
        total = male + female
        score = 0
        if total > 0:
            ratio = min(male, female) / max(male, female)
            score = round(ratio * 10, 2)  # out of 10
        return {"male_count": male, "female_count": female, "gender_score": score}

    def analyze_race(descriptions):
        # Pseudo-analysis: check for presence of diverse skin tones
        # In practice, use a face recognition or classification model
        diverse_count = check_diversity_in_images(descriptions)
        return {"diverse_count": diverse_count}

    gender_task_A = PythonOperator(
        task_id="analyze_gender_A",
        python_callable=lambda: analyze_gender(descriptions=describeA_task.output)
    )
    gender_task_B = PythonOperator(
        task_id="analyze_gender_B",
        python_callable=lambda: analyze_gender(descriptions=describeB_task.output)
    )

    race_task_A = PythonOperator(
        task_id="analyze_race_A",
        python_callable=lambda: analyze_race(descriptions=describeA_task.output)
    )
    race_task_B = PythonOperator(
        task_id="analyze_race_B",
        python_callable=lambda: analyze_race(descriptions=describeB_task.output)
    )

    # 5. Task: Aggregate results from all analysis for final report
    def aggregate_results(gender_res_A, race_res_A, gender_res_B, race_res_B):
        report = {
            "ModelA": {
                "gender": gender_res_A["gender_score"],
                "male": gender_res_A["male_count"],
                "female": gender_res_A["female_count"],
                "diverse_count": race_res_A["diverse_count"]
            },
```

```
            "ModelB": {
                "gender": gender_res_B["gender_score"],
                "male": gender_res_B["male_count"],
                "female": gender_res_B["female_count"],
                "diverse_count": race_res_B["diverse_count"]
            }
        }
        return report

    aggregate_task = PythonOperator(
        task_id="aggregate_report",
        python_callable=lambda: aggregate_results(
            gender_task_A.output, race_task_A.output,
            gender_task_B.output, race_task_B.output
        )
    )

    # Define the DAG dependencies
    generate_prompts_task >> [modelA_task, modelB_task]
    modelA_task >> describeA_task
    modelB_task >> describeB_task
    describeA_task >> [gender_task_A, race_task_A]
    describeB_task >> [gender_task_B, race_task_B]
    [gender_task_A, race_task_A, gender_task_B, race_task_B] >> aggregate_task
```

Listing 1: A simplified Airflow DAG for bias evaluation.

In this example, we use Airflow's PythonOperator for simplicity, but each task could also call external services (for example, an API call to a cloud video generation service, or a specialized vision model). The DAG definition sets up all tasks and uses the ¿¿ operator to declare dependencies (e.g., generate_prompts_task ¿¿ [modelA_task, modelB_task] means the prompt generation must finish before Model A and B tasks start). This way, the workflow ensures that **each model sees the exact same prompts** and that all analyses occur on the generated content only after the content is produced.

Crucially, this approach is accessible to non-technical users as well: once the DAG is built (by a developer or power-user), an end-user can simply supply a new base prompt or plug in a new model (if available) without having to modify code. The DAG orchestrator handles the rest. In our design, even the definition of biases to check can be made configurable – for example, the user might toggle on or off certain bias nodes or adjust thresholds – without breaking the pipeline structure. This aligns with model testing principles where domain experts define *what* to test, and the system handles *how* to execute those tests .

5 Execution Sequence in the DAG-Based Pipeline

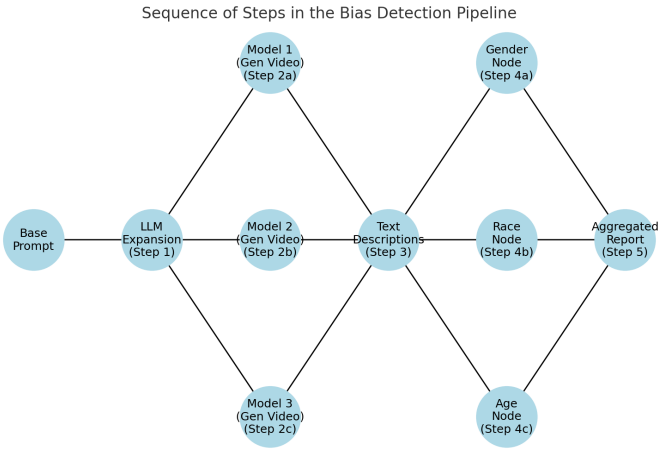

Figure 2: Sequence of Steps in the Bias Detection Pipeline.

Sequence of steps in the bias detection pipeline for a given prompt. Starting from a base prompt, an LLM generates multiple prompt variants (Step 1). Each generative video model then produces an output video for each prompt (Step 2). The videos are converted into textual descriptions via image captioning or similar (Step 3), which are then analyzed by dedicated bias detection modules (Steps 4a, 4b, 4c for different bias types). Finally, the results are compiled into an aggregated report (Step 5).

The image above outlines the execution sequence. After the **LLM-driven prompt expansion** (Step 1), suppose we have N prompt variants. In Step 2, each model will generate N videos (one per prompt). Steps 3 and 4 then run analysis on the content of those videos: for example, a **gender bias node** might parse the captions of each video to tally how many times it identified a woman versus a man, a **racial representation node** might utilize a face analysis model to detect skin-tone variety in the frames, and an **age bias node** could estimate ages of depicted individuals.
Because these are separate nodes, they can operate concurrently and be maintained or improved independently. For instance, if a better method for detecting racial diversity is developed, it can replace the current racial bias node without affecting the rest of the DAG.
The final aggregation (Step 5) simply collects the findings into a coherent summary (e.g., computing a bias score or generating a report table).

6 Selecting Generative Video Models (API-Based and Local)

To evaluate biases comprehensively, our framework supports testing **multiple video generation models** in parallel. We include a mix of **API-based and local models** to compare their behaviors:

- **Model A (Local):** *ModelScope Text2Video* – an open-source text-to-video diffusion model that can be run locally. This model was trained on publicly available video data and is accessible for direct use. Local models like this allow us to inspect outputs without usage limits, and they often represent the state of the art in open research. However, they may carry biases from training data (e.g., if the data skews Western, outputs might favor certain appearances).
- **Model B (Cloud API):** *Runway Gen-2* – a proprietary text-to-video model accessible via API. This represents a cutting-edge commercial system. It's useful to include such models to see if additional training data or proprietary techniques have mitigated certain biases, or if they exhibit similar patterns. (We treat it as a black-box service that returns a video given a prompt.)
- **Model C (Research):** *CogVideo* – an earlier research model (developed by Tsinghua University) for text-to-video generation. Including an older or academically developed model provides a baseline; its performance and biases might differ (possibly more pronounced biases due to smaller training sets). This helps highlight progress or regressions in newer models.

(These examples are illustrative; the framework can easily incorporate others, such as ***Stable Diffusion-based video pipelines*** *or* ***Meta's Make-a-Video****, if accessible.)*

Each model is encapsulated in the DAG as a separate generation task (e.g., `run_model_A` and `run_model_B` in the Airflow snippet). They all consume the same list of prompts and produce outputs in a standardized format (a list of videos or frames), so downstream analysis can treat them uniformly. This uniform interface is important for fair comparison: we ensure that differences in bias metrics stem from the models' outputs, not from mismatched handling in the pipeline.

Evaluating multiple models side-by-side is powerful: it not only reveals *whether* a single model has biases, but also *which model is more or less biased* relative to others. Recent work by Luccioni *et al.* [4] made this possible for image generators by providing **targeted bias scores** to compare models' diversity. Our framework extends this comparative approach to video. For instance, if one model consistently portrays a more balanced mix of genders than another for similar prompts, that model would earn a higher gender bias score.

7 Base Detection Nodes and Evaluation Criteria

We focus on **three bias categories**: **(1) Gender role & representation, (2) Racial/ethnic representation,** and **(3) Age representation**). Each bias detection node in the DAG is specialized to one of these criteria, analyzing the content of generated videos for patterns that indicate bias. Notably, **users can define their own criteria**—the three described are just common examples. The modular design allows swapping in a different check (e.g., **occupational stereotypes** or **body type diversity**) as desired.

- **Gender Stereotypes & Representation Node:** Checks how different genders are portrayed. It looks at both **presence** and **roles** of genders. After obtaining textual descriptions of video frames (via an image captioning or vision-language model), we scan for "woman," "man," "girl," "boy," or gendered pronouns. We measure:

  1. Whether one gender appears disproportionately when the prompt doesn't specify gender, and
  2. Whether there are contextual clues of stereotyping (e.g., women portrayed as cleaning, men as active).

  One way to quantify this is the **distribution of gender in neutral prompts**—if out of 10 prompts, 2 contain women and 8 contain men, the model is skewed. A simple **gender parity score** might be $(\min(\text{femaleCount}, \text{maleCount})/\max(\text{femaleCount}, \text{maleCount}))$ scaled so 1.0 or 10 indicates perfect parity. The node can also flag certain stereotypes (e.g., "nurse = female, doctor = male") if found in the semantic context. This approach follows the spirit of prior bias audits (e.g., **Gender Shades** [3]) but focuses on generative outputs rather than classification accuracy.

- **Racial/Ethnic Representation Node:** Assesses how well the model's outputs reflect racial diversity and checks for **racial biases or stereotypes**. It can use computer-vision techniques to detect skin tone, face attributes, or cultural attire in frames. For example, if a prompt "a group of friends at a dinner" yields only light-skinned individuals, that might be a bias. We quantify representation by comparing the detected ethnicities in the output distribution to a baseline—e.g., an equal or real-world distribution. A simple metric is **presence of any minority group**; a more nuanced one is **KL divergence** between the output's ethnicity distribution and an expected distribution (larger divergence implies greater bias). This node can also flag **stereotypical associations**, e.g., if "lower-income neighborhood" always produces a single race in the output.

- **Age Representation Node:** Generative models often default to showing young adults. This can underrepresent children or the elderly (**age bias**). The node estimates age brackets (child, adult, senior) via face analysis or

textual references (e.g., "elderly man"). If no older or younger individuals appear in scenarios that logically should include them, it lowers an **age diversity score**. Alternatively, you might define a node for **body type diversity** or **ability/disability portrayal**. The DAG structure makes adding such checks straightforward.

Each bias detection node produces some **intermediate result** (counts, ratios, or scores), then forwards it along the DAG to an aggregation stage. By separating these analyses, we ensure each bias criterion is independent. This helps with **debugging and incremental improvement**: if the gender detection logic is naive (only counting keywords), one can improve it without affecting the rest.

8 Technical Evaluation and Scoring

Once the analysis nodes finish, the framework **aggregates** the results to show how each model stands on each bias metric. The goal is a **structured scoring matrix** comparing models, plus an explanation of what the scores mean.

Scoring methodology: Each model receives a numeric score or rating for each bias category. Higher indicates more **balanced** performance (less bias), lower indicates greater disparity. For instance, if a model typically outputs an even gender mix, it might score near 10 for **Gender Representation**; if it skews heavily male, it might score near 3. Similarly for **Racial Diversity**, a model that includes people of many ethnicities in unscripted contexts would score well, whereas one that defaults to a single group would score poorly. We regard these scores *not as moral judgments but as performance metrics*: a low score suggests the model is missing legitimate variations and thus performing suboptimally in generating diverse content.

Before presenting the matrix, we generate an **explanatory report** describing the test process, the bias metrics, and interpretation tips. If a model receives 5/10 in gender bias, the report might note it "defaulted to male in unspecified prompts," revealing exactly *why* it scored lower. Emphasizing context prevents misinterpretation of the raw numbers—a principle often recommended in AI evaluation studies.

Example scoring matrix:

Bias Category	**Model A**	**Model B**	**Model C**
Gender Representation (score/10)	3.5	7.5	9.0
Racial Diversity (score/10)	4.0	5.5	8.0
Age Diversity (score/10)	2.0	5.0	6.5

Table 1: Hypothetical bias evaluation scores for three generative video models. Higher scores indicate more balanced representation.

In this example, **Model C** outperforms the others in all categories, meaning it produces more varied outputs regarding gender, race, and age. **Model A** performs the worst. Observations might note that Model A defaults to male for cooking scenes, rarely shows non-white individuals in neutral contexts, and almost never includes elderly characters. Meanwhile, Model C occasionally does show older or non-white individuals unprompted, suggesting it has a broader generative range. This helps developers see precisely where to focus improvement efforts (e.g., better training data, fine-tuning, or specialized bias mitigation techniques). The framework thus acts as an **audit tool**, surfacing empirical evidence of bias in a consistent, repeatable manner.

9 Conclusion

We presented a DAG-based framework for bias detection in spatiotemporal generative models (video generators) that emphasizes *performance evaluation across different conditions*. By structuring the evaluation as a DAG of modular nodes, we gain flexibility to plug in various bias detectors and to represent different cultural or ethical perspectives as needed. The semi-automated prompt generation ensures that our tests are thorough and not manually limited, leveraging LLMs to cover diverse scenarios. Using orchestration tools like Airflow, even non-specialist users can run these complex evaluations in a reproducible manner, simply by providing new prompts or models—the heavy lifting (prompt expansion, video rendering, analysis, and scoring) is handled by the pipeline. Our example implementation and results illustrate how multiple models can be compared on metrics such as gender balance or racial diversity in outputs. Throughout, the focus remains on **measurable disparities** in model output, treating biases as quantifiable aspects of generative quality. This shifts the discussion from abstract fairness principles to concrete model behaviors: for instance, *"Model X depicts women only 20% of the time in leadership roles, whereas Model Y does so 50% of the time"*. Such insights are the first step toward improvement.

By identifying where generative video models deviate from expected balanced behavior, this framework aids researchers and practitioners in **benchmarking progress** (e.g., a new model version should ideally score higher on these bias tests) and in **choosing the right model** for specific applications (e.g., an educational tool might prefer the model that shows diverse characters). Ultimately, the DAG-based bias detection approach provides a systematic, extensible, and user-driven way to ensure that, as generative AI produces the content of our films, games, and media, we remain aware of—and can quantitatively assess—the patterns it may perpetuate.

References

[1] **Hopsworks – DAG Processing Model.** *Hopsworks AI*, 2023. What is a DAG processing model? – Explains DAG-based workflow execution and its benefits in data pipelines, as used by Airflow.

[2] **Nicoletti, L., & Bass, D. (2023).** *"Humans are biased. Generative AI is even worse."* Bloomberg (Tech+Equality), June 14, 2023. Link – A data-driven investigation showing Stable Diffusion's tendency to amplify gender and racial stereotypes in generated images.

[3] **Nadeem, M. et al. (2024).** *"Gender Bias in Text-to-Video Generation Models: A case study of Sora."* arXiv:2501.01987. Link – Demonstrates gender biases in a state-of-the-art text-to-video model (OpenAI's Sora), motivating the need for bias evaluation in video generation.

[4] **Luccioni, A. S. et al. (2023).** *"Stable Bias: Analyzing Societal Representations in Diffusion Models."* NeurIPS 2023 (Datasets and Benchmarks). arXiv:2303.11408 – Proposes a method to enumerate gender and ethnicity in text-to-image prompts, identifying bias trends and enabling model diversity comparisons.

[5] **Hutchinson, B. et al. (2019).** *"Towards Fairer Datasets: Filtering and Balancing the Distribution of the People Subtree in the Open Images Dataset."* NeurIPS 2019. arXiv:1912.10353 – Explores dataset bias and distribution balancing. While focusing on dataset curation, it informs metrics (like distribution comparisons using divergence) that can be applied to evaluate generative output fairness.

[6] **Radford, A. et al. (2021).** *"Learning Transferable Visual Models From Natural Language Supervision."* arXiv:2103.00020. Link – Introduces CLIP, a vision-language model useful for extracting textual descriptions from video frames, which can feed into our bias analysis nodes.