# Safety Model Training and Integration

Cem Girgin

Email: girgin02@ads.uni-passau.de

***Abstract*—We evaluate the military-content safety filter embedded in the Group 6 object-counting application, with emphasis on how the model was trained, how it is integrated into the production pipeline, and how it performs. Evidence from project artefacts shows that the current safeguard functions almost exclusively as a keyword gate, despite being documented as a computer-vision model. The absence of trustworthy training data, meaningful evaluation metrics, and multimodal fusion has produced a brittle moderation layer that is easy to bypass and confusing for users. We distil the gaps in training, integration, performance, documentation, and user interface, and outline concrete technical steps to harden the safety pipeline.**

***Index Terms*—content moderation, safety filters, multimodal systems, responsible AI, user experience**

## I. Introduction

The Group 6 application combines an object-counting model (YOLOv8) with a bespoke "military vehicle" safety layer intended to prevent military imagery from being analysed. Executive stakeholders expect the safety model to reject uploads that could enable battlefield intelligence. During testing, however, we observed behaviour inconsistent with that mandate: the filter halts requests that contain certain keywords ("tank", "war") regardless of model confidence, yet the same images pass when the request text is altered. This report investigates the root causes across training, integration, and user-facing implementation.[1]

## II. Safety Model Training

### A. Training Data

Project files describe a model named `military_vehicle_detector_v1.0`, allegedly trained on a "Synthetic Military Vehicle Safety Dataset v1.0" with 1,000 images. No dataset artefacts or scripts were provided, and all runtime logs showed `is_military_detected = False` with `military_confidence = 0.0`. The absence of positive detections on clearly military imagery indicates either that the dataset never existed, was too small or homogeneous to generalise, or that the classifier outputs were ignored downstream. Trustworthiness is therefore questionable: synthetic-only data is prone to domain gaps, and without documentation we cannot verify geographic, temporal, or class diversity.

[1]**AI-Tool:** OpenAI ChatGPT prompt "Write a professional technical report evaluating an AI safety model that blocks military vehicle counting. Include sections on training, integration, performance, documentation, UX, and recommendations." returned an outline that informed the section ordering; the transcript is reproduced verbatim in the AI usage disclosure.

978-1-6654-1234-5/26/$31.00©2026 IEEE

Metadata embedded in the stale model card lists a 70/20/10 train/validation/test split and mentions class labels for "tank", "APC", and "jeep". Runtime inspection of the deployed weights, however, reveals only two output logits, reinforcing the suspicion that the documented dataset does not match the deployed artefact. Without provenance trails or sampling notes, there is no assurance that the data respects export-control laws or omits sensitive sources such as satellite imagery. The lack of demographic variation (e.g., vehicles from different nations, camouflage styles, or theatre conditions) further undermines representativeness. Consequently, any statistical claims about coverage are unverifiable.

### B. Training Setup

According to the existing model card, the detector follows a ResNet-like CNN trained with Adam for 50 epochs at batch size 32. There is no evidence of reproducible pipelines, hyperparameter sweeps, or checkpoints; no scripts were supplied to retrain or fine-tune with real imagery. The lack of reproducible implementation prevents the team from adapting thresholds or incorporating new data—a critical gap given the rapidly evolving landscape of military equipment.

Environment capture is also missing. No Dockerfile, Conda environment, or requirements list references the training job, so dependency drift is inevitable. GPU utilisation records are absent, leaving open questions about whether mixed precision or augmentation pipelines were enabled. In the absence of automated retraining hooks (for example, a scheduled CI job that rebuilds the detector when new samples are curated), the safety model cannot evolve in step with the main object counter.

### C. Potential for Re-Training

Although the architecture is nominally ResNet-based, the inference endpoint exposes a simple ONNX graph with a single dense layer on top of pre-computed embeddings. This simplifies re-training if the original intermediate features can be regenerated. A practical path would entail freezing the backbone, collecting labelled embeddings through data augmentation (occlusions, lighting shifts), and fine-tuning only the classification head. However, because class labels are hard-coded and the deployment bundle lacks versioning metadata, any new weights risk desynchronising with the FastAPI wrapper. Introducing model version tags and schema validation would be prerequisites before safe redeployment.

TABLE I
SAFETY GATE OUTCOMES FOR A CONSTANT TANK IMAGE UNDER VARIED LABELS

| Request | User Label | Backend Result |
|---|---|---|
| A | "tank" | Blocked; keyword "tank" logged; confidence 0.0 |
| B | "hardware" | Blocked; substring "war" matched; confidence 0.0 |
| C | "car" | Allowed; YOLOv8 returned zero detections |
| D | "person" | Allowed; YOLOv8 counted two people near the tank |

### D. Alignment with Model Card Guidance

Responsible AI artefacts recommend documenting purpose, data provenance, evaluation metrics, and limitations [1], [2], [4]. The current card claims strong metrics (95% accuracy, precision, recall) yet provides no confusion matrices or class names. Runtime behaviour contradicts these metrics, suggesting the numbers are either synthetic or stale. Ethical considerations (false positives on historical imagery, geopolitical bias) are mentioned superficially without mitigation plans.[2]

## III. Integration into the Application Pipeline

### A. Pipeline Placement

The safety component executes as a pre-processing gate before YOLOv8 inference. Server logs show the sequence: "♠ Running safety check" → either "Safety check passed" or "Safety check failed," followed by the counting model only when the gate passes. This placement is efficient—blocking early prevents wasted GPU cycles—yet it means any shortcomings in the safety gate directly determine policy enforcement.

### B. Trigger Logic

The gate relies on normalised text fields such as `object_type`. Deterministic tests produced the outcomes in Table I. The image classifier never contributes: even when blocking, the confidence remains zero. New integration code also exposes the matched keyword in the UI, confirming the string-match implementation.[3]

### C. Architecture Effects

Backend services lean on FastAPI. Integration added a safety module that executes before the object counter, short-circuiting the request when a forbidden term is found. There is no caching or cross-request memory: once an image is flagged, a simple label change by the user reopens the pipeline. No additional database schema or messaging queues were introduced, so the safety logic remains tightly coupled to the request handler—a maintenance risk.

[2]**AI-Tool:** SciSpace prompt "What information should a comprehensive model card include for an AI moderation model?" delivered the checklist reflected in this subsection; verbatim Q&A is captured in the disclosure PDF.

[3]**AI-Tool:** SciSpace prompt "What are known issues when a content moderation system analyzes images and text separately?" summarised literature on multimodal failure patterns that reinforced this observation; full response appears in the disclosure PDF.

### D. Pipeline Updates and Deployment History

Git history shows the safety filter landing in commit `8c1b9d5` as a minimal text guard appended to the existing request validator. Later releases adjusted the keyword list but did not alter control flow, so the guard still runs synchronously on the main API thread. The monitoring stack (Prometheus and Grafana) was never updated to track safety-specific metrics, and deployment manifests omit feature flags, meaning the filter cannot be toggled or A/B-tested without code changes. Trigger conditions therefore remain hard-coded, and rollback requires redeploying the service image.

## IV. Testing Methodology

Evaluation combined black-box experimentation with log inspection. We scripted a deterministic sequence using `mock_app.py` to submit the same tank image under varying labels, capturing API responses and server logs. Each run was timestamped to confirm that the safety check precedes model inference and to correlate UI modals with backend decisions. Log lines containing "Military keyword detected" and "Safety check failed" were extracted via `rg` to quantify trigger frequency. No experiment produced a non-zero confidence score, and no latency spikes were observed; timestamps confirmed sub-50 ms execution for the safety stage. While this approach validates behavioural claims, the absence of ground-truth labels or unit tests in the repository means results cannot be automatically regressed.

## V. Performance Assessment

### A. Observed Metrics

No instrumented metrics exist for sensitivity, specificity, or latency. The only available indicator is the log entry showing `blocked: True` with zero confidence. We therefore assessed performance qualitatively: false positives occur for benign words containing military substrings ("hardware"), while false negatives arise when users relabel the same image. Such imbalance (low precision and recall) mirrors failure patterns reported in moderation literature [5], [6].

### B. Failure Modes

- **Bypass Through Relabelling:** Changing `object_type` to "person" allowed analysis of soldiers standing next to a tank, defeating policy intent.
- **Spurious Blocks:** Words such as "hardware" trigger the substring "war." Similar collisions ("lawnmower", "steward") would be expected.
- **Zero-Confidence Decisions:** The classifier emits 0.0 confidence across all tests, indicating it is unused or misconfigured.

While latency overhead is minimal (string comparisons), reliability is inadequate for production policy enforcement.

TABLE II
EXPECTED VERSUS OBSERVED SAFETY SIGNALS

| Signal | Expected Behaviour | Observed Outcome |
|---|---|---|
| Military confidence score | High when disallowed content present | Remains 0.0 even as requests are blocked |
| Visual classifier contribution | Acts jointly with text heuristics | Never invoked; keyword gate alone drives decisions |
| Latency impact | Additional inference overhead | <50 ms string match regardless of image content |
| Relabelling attempts | Subsequent requests stay blocked | Relabelling to "person" or "car" bypasses safeguard |
| Audit telemetry | Logged probability, labels, decisions | Logs record keyword hit but omit quantitative evidence |

### C. Desired Metrics and Instrumentation Plan

To align with responsible deployment practices, the safety layer should expose quantitative metrics: true-positive rate on a hold-out set of labelled military scenes, false-positive rate on benign datasets (e.g., COCO civillian categories), mean decision latency, and the proportion of enforcement actions attributable to keywords versus vision signals. These metrics can be collected by wrapping the FastAPI endpoint with Prometheus counters (`safety_blocked_total`, `safety_bypass_attempts_total`) and by running nightly batch evaluations against curated validation corpora. Publishing these aggregates alongside release notes would create a feedback loop for policy owners and satisfy audit requirements.

### D. Observed vs Expected Signals

Table II juxtaposes the behaviour described in project artefacts with the expected safety signals. The gaps illustrate why the current deployment cannot satisfy the stated military-content policy.

## VI. Implementation and User Experience

### A. UI Workflow

The Flutter frontend surfaces a modal: "Safety Check Failed – This request was blocked by the safety system to prevent military vehicle counting," along with the offending keyword and "Confidence: 0.0%." Because neither onboarding flows nor documentation list disallowed content, users encounter unexpected blocks. Revealing raw confidence numbers contradicts the decision (blocking at 0%) and erodes trust.

### B. User Guidance and Recourse

No in-app guidance exists to help users comply, and there is no feedback or appeal mechanism. After a block, the upload form remains active, so users can immediately retry with a different label—encouraging experimentation to find loopholes. This undermines the policy and frustrates compliant users whose legitimate requests are blocked by substring collisions.

## VII. Limitations and Ethical Considerations

The current deployment violates several responsible AI guardrails:

- **Transparency:** Users are not informed about disallowed content categories, contravening the disclosure practices advocated by model card literature [1], [4].
- **Fairness:** Keyword heuristics may disproportionately block benign scenarios (museum exhibits, educational material) while failing to catch adversarial uses, resulting in unequal access.
- **Accountability:** Since confidence scores remain zero, engineering teams cannot audit whether the model ever contributes to decisions, impeding incident response.
- **Adaptability:** Lack of retraining pathways prevents incorporating new equipment classes or addressing emerging slang, leaving the filter stagnant in face of evolving threats.

These limitations emphasise the need for documented governance, continuous evaluation, and human oversight for high-risk requests.

## VIII. Operational Governance and Risk Register

Holistic integration requires controls beyond model accuracy. The following risk register captures the highest-impact failure modes observed during evaluation together with recommended mitigations.

**Bypass via relabelling** Enables hostile users to analyse military scenes by selecting neutral labels. Mitigate by caching image hashes, requiring joint text–vision consensus before passing, and rate-limiting repeated relabelling attempts.

**False positives on benign content** Blocks educational or historical imagery, undermining trust. Mitigate with word-boundary checks, a reviewed whitelist for common benign terms, and human-in-the-loop escalation for repeated triggers.

**Lack of audit trail** Prevents forensic reconstruction of moderation decisions. Mitigate with structured logging (request ID, image hash, signal scores) retained per compliance policy and surfaced in Grafana dashboards.

**Stale model weights** Allows drift away from real-world inventories. Mitigate with quarterly retraining cycles, dataset change logs, and regression suites that compare new weights against historical false-positive/negative cases.

**Operational blind spots** Leaves the organisation without observability into safety events. Mitigate by adding block-rate, latency, and anomaly-detection panels plus automated alerts when KPIs breach thresholds.

Policy governance should also define escalation paths: high-confidence detections could notify an on-call reviewer, whereas low-confidence cases might inform future dataset curation. Embedding the safety gate into the organisation's

incident response plan will ensure accountability when erroneous blocks or misses occur.

## IX. Recommendations

### A. Strengthen Training

- Gather a balanced dataset of real and synthetic military images with diverse viewpoints, weather, and equipment classes.
- Establish reproducible training scripts and checkpoints; support incremental fine-tuning when new data arrives.
- Benchmark with confusion matrices and ROC curves; publish metrics in the model card.

### B. Upgrade Integration

- Deploy a lightweight vision classifier (e.g., MobileNet) to score every upload; fuse text and vision signals with thresholds tuned via shadow testing.
- Cache safety outcomes per image (hash-based) to prevent relabelling bypasses within a session and store audit logs for high-risk outcomes.
- Maintain an allow/deny keyword list with boundary checks to avoid substring false positives; periodically review logs to retire noisy terms.

### C. Improve UI/UX and Documentation

- Announce policy constraints in onboarding or help content; disable disallowed labels in autocomplete.
- Replace the modal copy with a plain sentence such as "Request blocked: the system cannot process military equipment" and remove the contradictory "Confidence: 0.0%" line; add a help link that summarises allowed categories.
- Publish an updated model card covering purpose, data provenance, evaluation, limitations, and retraining cadence.

## X. Conclusion

The safety model currently operates as a rule-based keyword filter, yielding inconsistent enforcement of military-content restrictions. Missing or untrustworthy training data, lack of reproducible pipelines, and absence of multimodal fusion have left the gate vulnerable to trivial bypasses and false alarms. Integration within the application is shallow, providing neither architectural safeguards nor helpful user guidance. Addressing the identified gaps—particularly through real multimodal detection, calibrated thresholds, and transparent documentation—is necessary to align the system with responsible AI expectations and the project's policy goals.

## XI. AI Usage Disclosure

SciSpace (AI research assistant) was used to retrieve summaries of multimodal moderation challenges and model card best practices, informing the citations to [1], [2], [4], [6]. OpenAI ChatGPT supported draft structuring and language refinement; all generated suggestions were verified against observed system behaviour before inclusion. Prompts and outputs are documented separately in accordance with the AI Engineering Lab policy.


## Acknowledgment

Evaluation relied on internal backend logs and UI screenshots supplied by the development team, which substantiate the runtime behaviour described.


## References


[1] M. Mitchell *et al.*, "Model Cards for Model Reporting," in *Proc. Conf. Fairness, Accountability, and Transparency*, 2019, pp. 220–229.

[2] T. Gebru *et al.*, "Datasheets for Datasets," *Commun. ACM*, vol. 64, no. 12, pp. 86–92, 2021.

[3] Microsoft Azure Documentation, "What is Azure Content Moderator?," Microsoft Learn, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/content-moderator/overview

[4] Hugging Face, "Model Cards," 2023. [Online]. Available: https://huggingface.co/docs/hub/model-cards

[5] I. Toncheva, "Types of Content Moderation: Benefits, Challenges, and Use Cases," Imagga Blog, Feb. 2025. [Online]. Available: https://imagga.com/blog/types-of-content-moderation-benefits-challenges-and-use-cases

[6] V. U. Gongane, M. V. Munot, and A. D. Anuse, "Detection and Moderation of Detrimental Content on Social Media Platforms: Current Status and Future Directions," *Soc. Netw. Anal. Min.*, vol. 12, no. 1, p. 129, 2022.

[7] Project Group 6, "Backend Logs and UI Screenshots," internal communication, 2024.