CSCI 370 Final Report

CSM-Ricoh Vision

Aragorn Wang
Anya Streit
Matthew Jackson
Dmitry Weakly

Revised June 15, 2025

CSCI 370 Summer 2025

Professor Kristen Peglow

Table 1: Revision history

Revision	Date	Comments
New	May 18, 2025	Wrote the first draft of the introduction, functional requirements, non-functional requirements, risks, and definition of done sections.
Rev – 2	May 27, 2025	Updated the introduction, functional requirements, non-functional requirements, risks, and definition of done sections. Created the first draft of the system architecture, team profile, references, and appendix A sections.
Rev – 3	June 3, 2025	Updated the introduction, functional requirements, non-functional requirements, risks, definition of done, system architecture, references, and appendix A sections. Created the first draft of the software test and quality and ethical considerations sections.
Rev – 4	June 8, 2025	Updated all previously existing sections and created the first draft of the results, future work, and lessons learned sections. First significant test results added. Added much more content to the references and appendix A sections.
Rev – 5	June 15, 2025	Updated all previously existing sections and added more test results. Finalized the acknowledgements section.

Table of Contents

I. Introduction

High-level Description

Over five weeks, our team developed a pipeline that uses NVIDIA Isaac Sim (Replicator) and Pixar's Open Universal Scene Description (USD) to generate labeled RGB-D video datasets (i.e. each RGB frame is a realistic rendering of our factory scene of interest, with corresponding bounding boxes for each detected paperstack object). We then trained state-of-the-art object detectors (YOLOv11) on synthetic data, evaluated them against real-world captures, and demonstrated real-time inference on Jetson AGX Orin or NVIDIA RTX 6000 Ada 48 GB hardware mounted on Ricoh's pick-and-place robot.

Client

Ricoh is a Japanese multinational company primarily known for its imaging and electronics products, including printers, copiers, and other office equipment. Ricoh desires this project to help advance their new robotics division that's developing pick-and-place robots to help automate product assembly lines.

Source of Data

We generated synthetic physical-world video data via Cosmos World Foundation Models, Isaac Sim Replicator, and OpenUSD. Essentially, Isaac Sim Replicator is what we use to create a virtual mock-up of the physical environment

(paperstacks, conveyor belt, robotic arm, etc) and be able to programmatically vary the environment's parameters (camera angle, lighting exposure, robotic arm texture, etc). OpenUSD is the file format that all the environment's assets and object interactions are encoded in. Cosmos World Foundation Models are the transformer-diffusion image-generation models (think OpenAI Sora or DALL-E) that we use to "inpaint" the Isaac Sim RGB frames (each frame also has depth, edge, and semantic segmentation [what pixel regions correspond to what object classes]) data) to look more realistically textured and lit.

Dataset and Objects

In order to support our video dataset generation efforts, we referenced real-world captures of paper stack or paper stack-like objects (including books or other printed media) to serve as a guideline for what our synthetic data would look like. These objects are also what our object detector models were trained to detect.

Hardware Interface

- Training/editing: Desktop with NVIDIA RTX 6000 Ada 48 GB and NVIDIA Titan RTX GPU, or Ricoh provided AWS Marketplace Virtual workstation for Isaac Sim
- Inference prototype: Jetson AGX Orin (embedded) and NVIDIA RTX 6000 Ada 48 GB (workstation)
- Robotic platform: Universal Robots Cobot arm at Ricoh's Boulder lab

Stakeholders of Software

- Ricoh Robotics Group (ATI): project sponsor, will integrate into their automation stack
- CSCI 370 students & Professor: academic advisors, code reviewers
- End users: assembly-line technicians at Ricoh's customer at their print shops

Who Maintains the Software?

After project hand-off, Ricoh's Robotics Group (ATI) will maintain the simulation scripts, training pipelines, and inference modules in their internal GitLab repository

II. Functional Requirements

1. Generate at least 10,000 labeled RGB frames with varied lighting, backgrounds, and paperstack poses
2. (Stretch goal) Generate an additional 10,000 labeled RGB frames with varied lighting, backgrounds, and paperstack poses
3. (Stretch goal) Use transfer learning to train semantic segmentation detectors to at least 85% Intersection over Union (IoU) on a held-out synthetic validation set
4. (Stretch goal) Evaluate detectors on real captures: Report IoU and inference latency
5. (Stretch goal) Deploy best-performing model to run with at least 10 FPS on inference hardware with at most 65 ms per frame latency

III. Non-Functional Requirements

1. Source code follows autopep8 style and uses linters like pylint or pylance
2. Code and artifacts version controlled by git-lfs so data and model artifacts are traceable
3. Sufficient documentation for project continuance by future contributors and maintainers
4. Store any platform or API (e.g. AWS) credentials in environment variables; never hard-code secrets

IV. Risks

Risk 1 — Domain Gap

Likelihood: Very likely
Impact: Major

Mitigation Plan: Use extensive domain randomization, introduce real-world texture overlays, and fine-tune on small real dataset

Risk 2 — Hardware Bottleneck

Likelihood: Likely
Impact: Moderate
Mitigation Plan: Profile with NVIDIA Nsight; optimize model (pruning and ablation analyses) to meet latency targets

Risk 3 — Simulation Collision Errors

Likelihood: Unlikely
Impact: Minor
Mitigation Plan: Add sanity checks in USD scripts and scene physics settings; log and auto-retry failed simulation frames

Risk 4 — Real-World Sensor & Lighting Mismatch

Likelihood: Very likely
Impact: Major
Mitigation Plan: Augment synthetic data with photometric variations (e.g. HDRI environment maps, specular and shadow noise) and capture a small "pilot set" of real RGB-D frames under diverse lighting and use them to fine-tune or calibrate the detector

V. Definition of Done

- Dataset: At least 10,000 labeled frames published in Ricoh's cloud bucket
- Codebase: Fully documented GitLab repo with README, API docs, and examples
- Delivery: Zip file of artifacts delivered via Ricoh's internal file share by Week 5 end
- (stretch component) Models: Trained YOLOv11 weights stored, with training logs
- (stretch component) Evaluation Report: PDF comparing synthetic vs. real performance metrics
- (stretch component) Demo: Real-time inference running on Orin-mounted Universal Robots (UR) arm in Boulder lab

VI. Model and Dataflow

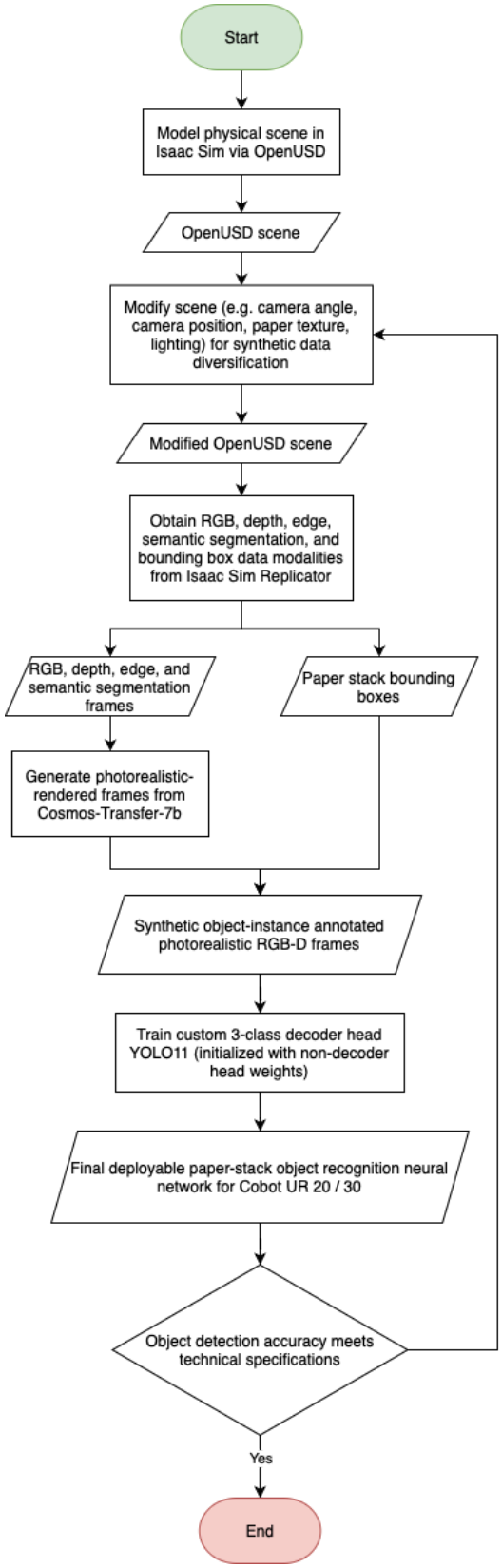

Figure 1: Synthetic dataset and model training flowchart

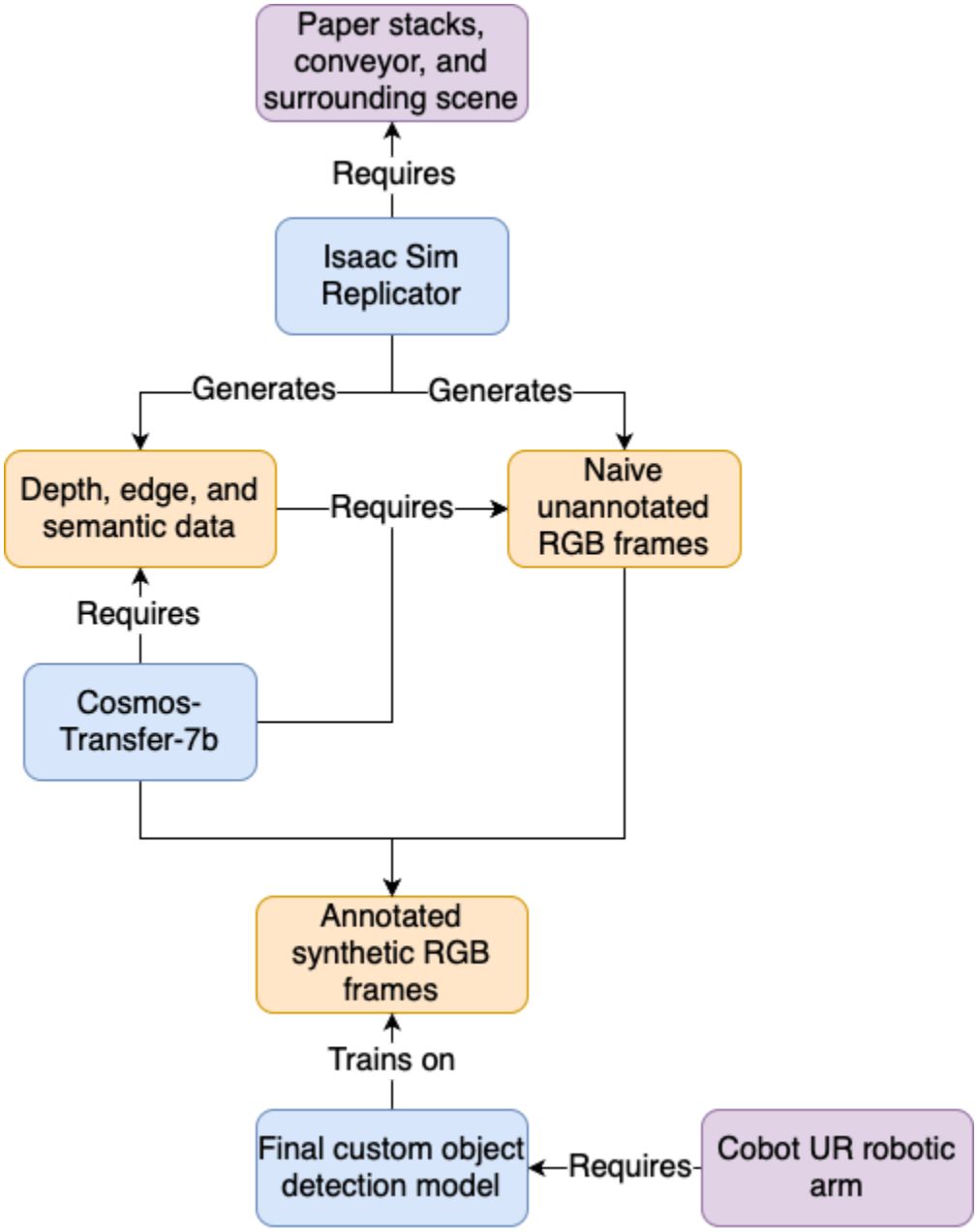

Figure 2: Component interaction hierarchy (annotator object detection model is AKA labeling object detection model, production object detection model is AKA inference object detection model)

Our team ran across a number of technical design problems during the Ricoh 2 project that slowed down development speed and influenced general project delivery. The main difficulties we tackled are summarized here:

1. Insufficient GPU and VRAM Resources

   For our four-member team, neither Ricoh nor the Colorado School of Mines were able to provide sufficiently GPU-equipped workstations for a period of time. Only one workstation—albeit hardly meeting the minimum criteria—was able to run inference on the diffusion transformer Cosmos-Transfer models. This resulted in a critical bottleneck: only one engineer was able to iterate on model development or testing at a time, causing delayed feedback cycles (detailed in figure 1 and 2) and idle times for other team members.

2. Limited Hardware for Isaac Sim Workflows

   The hardware limitations also applied to simulations of robotics. To create and label OpenUSD scenes, we had a single machine that could launch Isaac Sim and run the Replicator annotator workflows. Our synthetic data pipeline was therefore limited in scalability, which necessitated manually queuing simulation jobs and lengthened the turnaround time for augmenting training data.

3. Model Reliability: Excessive Hallucination in Cosmos-Transfer1-7b and Cosmos-Predict1-14b

   We tested the Cosmos-Transfer1-7b and Cosmos-Predict1-14b variants as a direct "real video sample to synthetic video" generator and found excessive amounts of artifacting and visual hallucinations. Often, the diffusion backbone adds elements that are completely different from the original footage, making the outputs unsuitable for tasks like object detection or domain adaptation. Upon looking into different conditioning techniques and denoising schedules, we developed a hybrid strategy—such as coupling with explicit physics-based rendering.

4. Jira Configuration and Workflow Separation

   Due to a misconfigured Jira board at the beginning of the project, it was challenging to differentiate between tasks for the Ricoh 1 and Ricoh 2 teams. Sprint planning was hampered by this misalignment, which also resulted in duplicate tickets and unclear ownership. Since then, we reorganized the board with distinct projects and tailored processes, but as team members got used to the new setup, lingering misunderstandings about ticket assignment and reporting still existed.

5. Dependency and Environment Management

   Version conflicts had been common due to the project's heterogeneous stack, which included CUDA toolkits, Python ML frameworks (PyTorch, TensorFlow), ROS 2, and numerous visualization libraries. CUDA driver updates, in particular, had occasionally caused incompatibilities with current conda environments, necessitating laborious rollbacks or container rebuilds. To reduce this risk, we standardized Docker-based development environments with locked dependency versions.

VII. Software Test and Quality

This section describes the planned testing activities that ensure each functional and nonfunctional requirement is met. For every requirement, there is a corresponding test plan that includes the purpose of the test, a description of how it was executed, the tools required, the pass/fail threshold (linked back to the requirement), relevant edge cases, and how results were recorded.

Test Plans for Functional Requirements

FR-1: Generate at least 10,000 labeled RGB frames with varied lighting, backgrounds, and paper-stack poses

- Purpose:
  - Verify that the synthetic-data pipeline produces the required number of frames and that labels are correct and diverse.
- Description:
  - Run the Isaac Sim + OpenUSD pipeline to generate a batch of frames.
  - Count output png files and corresponding label files.
  - Visually inspect a random 5% sample for correct pixel-wise labels.
  - Automatically verify that metadata (e.g. lighting parameters, pose) varies according to randomization settings.
- Tools Used:
  - NVIDIA Isaac Sim (headless mode)
  - Custom Python script (e.g. scripts/verify_data.py)

        - Shell utilities (e.g. ls, wc) to count files
- Acceptability Criteria:
    - At least 10,000 frames must exist.
    - At least 95% of randomly sampled images must have correct labels (verified via script comparing label mask against known ground-truth for simple synthetic scenes).
- Edge Cases:
    - Low contrast lighting conditions (e.g. dark ambient + paper textures)
    - Paper partially occluded by other geometry (overlapping stacks)
    - Background textures very similar in color to paper
- Testing Results:
    - Counts of generated frames and labels
    - Labels accuracy percentage from sampling
    - Metadata variation report (lighting and pose statistics)

FR-2 (Stretch): Generate an additional 10,000 frames (total of at least 20,000)

- Purpose:
    - Confirm that the pipeline can be scaled to produce at least 20,000 frames without duplicates and maintaining label accuracy.
- Description:
    - Repeat the FR-1 process with a different randomization seed to ensure no repeated frames.
    - Count the total number of frames and labels.
    - Perform the same sampling inspection and metadata checks.
- Tools Used:
    - Same as FR-1
- Acceptability Criteria:
    - Total of at least 20,000 frames
    - Label-accuracy criteria same as FR-1
- Edge Cases:
    - Check for accidental duplication of identical RGB + label pairs
    - Scenes with extreme lighting (near-overexposed)
- Testing Results:
    - Counts of generated frames and label
    - Label accuracy percentage
    - Duplication check summary

FR-3 (Stretch): Train object detector to achieve at least 85% IoU on held-out synthetic validation set

- Purpose:
    - Ensure the model architecture and training hyperparameters yield the target mean Intersection over Union .
- Description:
    - Fine-tune a chosen segmentation backbone (e.g. YOLOv11) on the synthetic training set.
    - Run validation on the held-out synthetic set (e.g. 2,000 images).
- Tools Used:
    - PyTorch (at least 2.6.0) with CUDA support
    - Custom training script (e.g. train/train_segmentation.py)
    - PyTorch metrics or mmsegmentation evaluation module
    - NVIDIA Nsight for performance profiling (optional)
- Acceptability Criteria:
    - IoU of at least 85%

- Edge Cases:
    - Synthetic scenes with rare combinations (e.g. paper partially out of frame, extreme tilt)
    - Adversarial textures in the background (e.g. background resembling paper)
- Testing Results:
    - IoU values per class and overall
    - Training and validation loss curves
    - Any notes on convergence issues

FR-4 (Stretch): Evaluate detectors on real captures, reporting IoU and inference latency

- Purpose:
    - Ensure that models generalize to real data with acceptable accuracy and latency.
- Description:
    - Capture 500 real RGB-D frames in Ricoh's Boulder lab under varied lighting conditions.
    - Run the trained model on these frames.
    - Calculate IoU against hand-annotated ground truth.
    - Measure per-frame inference time on RTX 6000 workstation and Jetson AGX Orin embedded platform.
- Tools Used:
    - Hand-annotation tool (e.g. LabelMe) for creating ground truth
    - Python evaluation script (e.g. eval/evaluate_real.py) using PyTorch
    - Python time module or torch.cuda.Event for measuring latency
- Acceptability Criteria:
    - IoU greater than or equal to 70% overall on real captures
    - Per-frame inference of, at most, 65 ms on RTX 6000 and, at most, 80 ms on Jetson AGX Orin
- Edge Cases:
    - Challenging lighting conditions (e.g. harsh shadows, reflections)
    - Partial occlusion of paper by robot arm
    - Background clutter unseen in synthetic data (e.g. tools, cables)
- Testing Results:
    - Real-data IoU per class and overall
    - Latency statistics (i.e. average, min, max)
    - Notes on failure cases (e.g. classes with low IoU)

FR-5 (Stretch): Deploy best-performing model to achieve at least 10 FPS on target inference hardware, with at most 65 ms per frame latency

- Purpose:
    - Validate that the optimized deployment pipeline meets throughput requirements on both embedded and workstation hardware.
- Description:
    - Export the PyTorch model to TensorRT (for Jetson AGX Orin) and ONNX (for RTX 6000).
    - Run inference on 1,000 sequential frames, measuring average frames per second.
    - Verify that batch size 1 yields at least 10 FPS.
- Tools Used:
    - NVIDIA TensorRT for Orin optimization
    - ONNX Runtime for RTX 6000
    - Docker container configured with required drivers and SDKs
    - Custom benchmarking script (e.g. deploy/benchmark.py)
- Acceptability Criteria:
    - Sustained at least 10 FPS (at most 100 ms per frame) on both devices
    - Measured latency of, at most, 65 ms on RTX 6000, and, at most, 100 ms on Orin

- Edge Cases:
    - Worst-case scene resolution (e.g. very large images)
    - Thermal throttling scenarios (e.g. long continuous runs)
- Testing Results:
    - Average FPS and latency per device
    - Any notes on throughput degradation over time

Test Plans for Non-Functional Requirements

NFR-1: Code must follow autopep8 style and pass pylint checks

- Purpose:
    - Enforce code consistency and readability according to project guidelines.
- Description:
    - Run autopep8 --in-place --aggressive on all .py files.
    - Execute pylint across the entire src/ directory.
    - Review linter warnings and errors, fix any with severity above “convention.”
- Tools Used:
    - autopep8 (version at least 1.6)
    - pylint (version at least 2.15)
- Acceptability Criteria:
    - Zero pylint errors classified as “error” or “fatal”
    - At most five “warning” messages
    - All “convention” and “refactor” messages must be addressed
- Edge Cases:
    - Newly added modules missing docstrings
    - Very large files (500 lines or more) accidentally skipped by linters
- Testing Results:
    - Linter report summary and list of remaining warnings (if any) and how they will be addressed

NFR-2: All code and artifacts must be version controlled with Git-LFS for large files

- Purpose:
    - Ensure reproducibility and traceability of data artifacts and models.
- Description:
    - Verify presence of .gitattributes with patterns for large files (e.,g. .pt, .pth, .png).
    - Check that no large binary (50 MB or greater) exists untracked by LFS by running git lfs ls-files and comparing to git ls-tree.
- Tools Used:
    - Git (version at least 2.49.0) with Git-LFS (version at least 3.6)
    - git-lfs ls-files command
- Acceptability Criteria:
    - 100% of model artifacts and generated datasets are tracked by LFS
    - No file greater than 50 MB appears in regular Git history
- Edge Cases:
    - Uncommitted large files present in the working directory
    - Misconfigured gitattributes missing a new file pattern
- Testing Results:
    - Git-LFS tracking report
    - List of any untracked large files and corrective actions

NFR-3: Sufficient documentation for future contributors, including README and API docs

- Purpose:
  - Confirm that documentation exists, is up to date, and covers setup and maintenance.
- Description:
  - Check for docs/README.md, docs/API.md, and inline docstrings in all modules.
  - Attempt to build HTML docs via Sphinx or MkDocs (if configured).
  - Have a teammate perform a fresh clone and follow setup instructions, noting any gaps.
- Tools Used:
  - Markdown files in docs/ directory;
  - Sphinx (version at least 8.2.3) or MkDocs (version at least 1.6.0), if configured
  - Documentation completeness checklist
- Acceptability Criteria:
  - 100% of public functions and classes must have docstrings
  - README must include project overview, installation steps, usage examples, and contact information
- Edge Cases:
  - Instructions referencing OS packages not installed (e.g. CUDA toolkit is missing)
  - Outdated dependencies listed in docs
- Testing Results:
  - Documentation build logs
  - Feedback from teammates on any missing or unclear steps

NFR-4: Store platform/API credentials in environment variables; no hard-coded secrets

- Purpose:
  - Prevent accidental exposure of sensitive information.
- Description:
  - Grep the repository for keywords: API_KEY, PASSWORD, TOKEN, etc.
  - Run git-secrets or similar tools to detect accidental AWS keys or other patterns.
  - Confirm that any code accessing credentials refers only to os.environ.
- Tools Used:
  - grep or rg (ripgrep) for suspicious patterns
  - git-secrets with common regexes configured
  - Code review by at least two teammates
- Acceptability Criteria:
  - Zero hard-coded secrets (no plaintext credentials found)
  - All credential access uses os.environ.get() or equivalent
- Edge Cases:
  - Environment-variable keys misspelled (e.g. API KEY vs. API_KEY)
  - Older commits containing keys not yet purged from history
- Testing Results:
  - Findings from grep and git-secrets
  - Any code review notes on credential usage

Overall Edge Cases

We included the following scenarios to ensure robustness:

- Extremes in lighting and texture, such as completely homogeneous or highly contrasted lighting (pure black backgrounds), are used to test segmentation in unreal lighting conditions.
- To assess the robustness of the model against annotation errors, introduce up to 5% of incorrectly labeled pixels into the synthetic training set.

- By executing inference benchmarks on a Jetson AGX Orin in a closed setting with a fully charged battery and tracking performance over lengthy sequences (500 frames or more), hardware throttling is demonstrated.
- By simulating new environments with Docker containers and confirming that dependency versions are pinned in requirements.txt or the environment, version conflicts can be avoided. YML generates conflict-free, identical configurations.

Overall Test Results

- A centralized test_results/ directory within the GitLab repository will receive the output of all test executions, whether they are scripted or manual.
- A timestamped log file (such as results_FR-1_20250601.log) will be generated by each test script.
- Any failed tests will be reviewed at a weekly "Quality Check" meeting, and Jira tickets will be created to track fixes. A requirement will be marked as complete in the Definition of Done once all tests have passed and the logs have been approved.
- The appendix of the final report will contain confusion matrices and per-class IoU tables for the model-accuracy tests (FR-3 and FR-4). A retraining or data-augmentation iteration will be initiated if thresholds are not met.

VIII. Project Ethical Considerations

In creating a synthetic-data pipeline and running object-detection models on robotic hardware, our team identifies several ethical aspects. We refer our analysis to the ACM Code of Ethics and Professional Conduct (e.g. "1.2 Avoid Harm," "1.4 Be Fair and Take Action Not to Discriminate," "2.5 Give Proper Credit for Intellectual Property," "3.1 Ensure Privacy," and "4.2 Articulate and Address Ethical Concerns") and the IEEE Code of Ethics (e.g. "1. To hold paramount the safety, health, and welfare of the public," "3. To be honest and realistic in stating claims or estimates," and "5. To avoid injuring others, their property, reputation, or employment by false or malicious action"). We can identify four key considerations:

1. **Safety and Reliability of Deployed Systems**
   ACM 1.2: Avoid Harm. As our object detectors will eventually drive pick-and-place robotic arms interacting with human technicians, robustness of models is of paramount importance. Synthetic training can introduce distributional shifts when the robot perceives unmodeled real-world conditions (e.g. unmodeled occlusions or reflections). To satisfy ACM 1.2 and IEEE 1, we commit to strict verification on real captures and continuous monitoring of inference logs. Misclassification by any model of "paperstack" objects may lead to misunderstanding or collisions. Fail-safe automated checks will therefore be introduced: if below some conservative threshold, confidence, the robot must go into an error state rather than risk unintended motion.

   ACM 3.3: Understand and Obey Laws. As robotic operations can cause bodily injury or property loss, we will have our system adhere to applicable safety standards (e.g. ISO 10218 for industrial robots). We must document all instances where model drift and sensor mismatch can violate local safety regulations to ensure transparency and accountability are upheld (IEEE 3).

2. **Data Integrity, Privacy, and Transparency**
   ACM 3.1: Maintain Privacy. Although our synthetic data is not required to record personally identifiable information, any "pilot set" of real RGB-D captures will have to be sanitized to keep from capturing human faces or trade-secret package designs by accident. All real-world imagery will be filtered and anonymized before it is released outside our organization. In line with IEEE 1 and ACM 4.2 ("Articulate Ethical Considerations of New Systems"), we will release a public README detailing dataset creation protocols, potential biases resulting from domain randomization, and known limitations to enable downstream users to make well-informed decisions.

   ACM 2.5: Give Due Intellectual Property Credit. Some components (e.g. NVIDIA Replicator, Cosmos World Foundation Models, YOLOv11) are proprietary or open-source software. We will have transparent acknowledgments in our documentation and, where applicable, respect licensing terms (e.g. provide attribution

or link back to original repositories). Such transparency aligns with IEEE 3, which requires honesty about system capabilities and external dependencies.

3. **Fairness and Mitigation of Algorithmic Bias**
   ACM 1.4: Act Fairly and Avoid Discriminating. While our target class (“paperstack”) is not human, discrimination can occur when synthetic lighting or texture conditions over-sample certain circumstances (e.g. bright lab vs. dark factory). If real-world lighting conditions are biased toward disproportionately inducing detection failure, this would unfairly bias against certain operational contexts (e.g. night shift assembly lines). In response, our randomization-to-domain strategy will intentionally sample across extensive ranges of lighting and background variations. We will also measure performance differences across subsets (e.g., overhead vs. side lighting) and adjust our synthetic pipeline accordingly until accuracy differences are reasonable tolerance (best case 5% gap) under ACM 1.4.

4. **Environmental Impact and Responsible Resource Usage**
   IEEE 7: To seek, accept, and offer constructive criticism of technical work. ACM 1.6: Respect Other People's Privacy. Utilizing large model training on GPUs or cloud infrastructure is power-hungry. To best satisfy this principle, we intentionally selected the lowest CO_2-emitting compute nodes and chose model architectures (e.g. less power-hungry transformer versions) with good performance but little training time. Upon resource constraints requiring longer simulation runs, we batched experiments for off-hour running (which are usually hours that data centers will normally use cleaner energy) and carefully documented these decisions so future teams can criticize or build upon our approach.

Finally, we commit to maintaining open channels of communication with Ricoh's Robotics Group, peer CSCI 370 students, and assembly-line workers. Any known ethical concerns, such as limitations on generalization to new factory settings or potential mechanical failure, will be unambiguously established in hand-off documentation. By integrating ACM and IEEE principles into our design, testing, and deployment processes, we promise to not only enhance Ricoh's robotics but also ensure the highest safety, fairness, transparency, and environmental responsibility standards.

IX. Project Completion Status

Performance Testing Results

FR-1: Generate at least 10,000 labeled RGB frames with varied lighting, backgrounds, and paper-stack poses

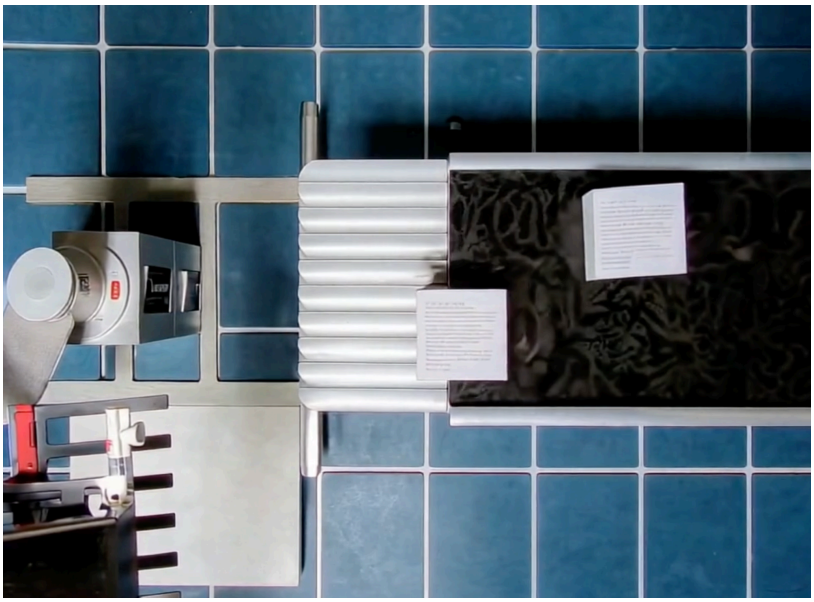

Figure 3: Sample frame from test1 video

- We have been able to generate enough frames to meet the 10,000 goal, but we only recommend keeping the 6,600 frames that we've deemed as being high-quality and worthwhile in the Github.
- To fully meet FR-1, the remaining 3,400 frames implicitly require about 23 additional videos of same or greater duration to the existing videos.
- The robot arm mechanics seen in the sample video output above (shown in figure 3) are somewhat wobbly and possibly exaggerates the unstableness of the UR Cobot's custom gripper.
- The rendered conveyor belt's texture and lighting exhibits hallucinatory blurs and liquid-like swirling regions (visible in figure 3's conveyor belt) that are entirely inconsistent with the belt's kinematics.
- However, the paper stack kinematics, overall scene lighting, and general texture realism are rendered quite well.

Usability and Maintainability Testing Results

NFR-1: Code must follow autopep8 style and pass pylint checks

- Every single python file (only .py's, all .ipynb's have been converted to .py scripts) 100% adheres to autopep8 style standards, with zero violations.
- All modules and functions have docstrings, proper 4-space indentation, no trailing whitespaces, etc.

NFR-2: All code and artifacts must be version controlled with Git-LFS for large files

- All cosmos-transfer checkpoint files in build/checkpoints/ are tracked.
- All generated synthetic video and JSON files (currently unlabeled) recursively located in video-assets/ are tracked.
- All converted dataset files (LVIS and OpenImagesV7) for inference YOLOv11 training recursively located in data/converted_lvis/ and data/converted_openimagesv7/ are tracked.
- All necessary large code and artifact files and directories are recursively tracked via Git-LFS.

NFR-3: Sufficient documentation for future contributors, including README and API docs

- The simple environment setup instructions (simply running three non-interactive installation scripts we've created) are detailed in the nicely formatted markdown README.md file.
- The documentation for understanding the source code more deeply and making modifications to our pipeline-stage scripts (i.e. scene generation, synthetic dataset generation, and inference object detection model training and deployment) is still a work-in-progress since the MVP of the project is still incomplete.
- All API docs are no longer necessary for the user due to our detailed commenting describing our usage of the more obscure functions from niche Python and Conda packages.

NFR-4: Store platform/API credentials in environment variables; no hard-coded secrets

- This actually turned out to be irrelevant, since none of our packages' APIs require user-registered credentials.

Testing Summary

Overall, our test summary shows that all the non-functional requirements have been fully satisfied or found not to apply. All pylint and autopep8 tests were successful with zero errors, all crucial artifacts are traced through Git-LFS, and no hard-coded credentials exist while the core functional tests remain unfinished. Up to this point, only 6,600 of the required 10,000 labeled frames have been created, causing wobble in the UR cobot's patented gripper and producing hallucinated texture patches on the fake conveyor belt. Therefore, the remaining 3,400 frames along with all stretch-goal evaluations, including large-scale frame generation, model accuracy on synthetic and real data, and deployment performance goals, are yet to receive increased GPU compute power and time. Our test environment and timeline are ready, and with enough hardware availability, we will execute the pending tests to meet the project completion requirements.

Features Currently Incomplete

Stretch features and test results for FR-2, FR-3, FR-4, and FR-5 are currently incomplete. However, all of the actual scripts and software engineering required for all of them are complete. The reason for this seemingly large lack of progress is due to having heavily insufficient hardware for the first three weeks of this field session. Thus, we simply require more compute time on our tier 1 (NVIDIA RTX Ada 6000) and tier 2 (Google Cloud Platform GPU-compute instance) to generate the (ideally) 20,000 desired labeled synthetic RGB frames, achieve 85% minimum IoU for inference object detection models, and checkpoint files necessary to deploy the inference object detection model on the NVIDIA Jetson AGX Orin 48GB platform.

X. Future Work

As referenced in Section IX and taking into account the remaining stretch-goal targets (i.e., scaling up our dataset generation to a complete 20,000 frames, achieving at least 85% IoU on synthetic validation, validating on real captures with acceptable latency, and deploying the optimized model with a minimum of 10 FPS on target hardware) our next actions will be to add more GPU resources as well as automating our simulation pipeline to eliminate manual queuing, executing all outstanding dataset and evaluation runs, and adding hybrid physics-based rendering techniques in an effort to reduce visual artifacts in synthetic scenes. In addition, we will include ongoing integration features to address end-to-end performance and regression testing, create full hand-off documentation and API documentation for future contributors, and execute carbon-aware scheduling of off-peak training activities to minimize environmental impact. Finally, we aspire to develop a continuous refinement cycle between the object detector and the UR cobot controller to verify real-world assembly-line performance and safety before official hand-off.

XI. Lessons Learned

Over the course of the past 5 weeks, our team learned several lessons through the challenges and hardships we faced as well as the experience gained from working with the project in general. The most important of these lessons are summarized as follows:

1. Hardware Availability is a Must

One issue we consistently faced was a lack of hardware resources. We therefore learned how critical it is to both determine what our hardware requirements are and then secure the necessary hardware required to do our task. Doing so will ensure that our progress is not bottlenecked by acquisition and will result in less idle time. Creating contingency plans in the event that more hardware is necessary is also something that we learned is important to do.

2. Parallelization Vastly Improves Team Efficiency

Our team's ability to work in parallel was limited by hardware requirements but also more importantly by planning. Upon having team members be responsible for individual parts of the pipeline, much more work was done. Thus, having more of the project planned and tasks made more granular early on, more tasks can be completed in the same amount of time.

3. Stretch Goals Require Early Planning

Building off of the last point, while our stretch goals like real-world validation were within reach, delays in our production pushed them out of our timeline. By accounting for these targets in planning or preparing for them in parallel could allow for them to be achieved more timely.

XII. Acknowledgments

We would like to thank the Ricoh employees and consultants who provided us with this amazing, cutting-edge project: Ivan Portilla, Ziling Zhang, and Jay Patel. They were extremely kind, supportive, and inspiring in their collaboration and supervision of our work.

We would also like to thank the Mines IT department for assisting us with our tier 2 Google Cloud Platform access approval and access.

And of course, we would like to thank our faculty advisor, Kristen Peglow, for her wonderful guidance throughout this difficult yet rewarding experience.

And finally, we would like to especially thank Professor Kathleen Kelly for her extremely gracious financial generosity, her responsiveness to our communications, and her understanding in regards to our hardware acquisition difficulties and understanding of course expectations.

XIII. Team Profile

	**Name:** Matthew Jackson
	Academic Standing: Incoming fourth-year undergraduate
	Fields of Study: Computer Science (data science track), Electrical Engineering (Information and System Sciences Track)
	Hometown: Round Rock, TX
	Work Experience: Actian Corp.
	Extracurriculars: Jazz Piano/Trumpet, Personal Programming, Hockey, Skiing, Mountain Biking, Cooking
	**Name:** Dmitry Weakly
	Academic Standing: Incoming third-year undergraduate
	Fields of Study: Computer Science (general track)
	Hometown: Denver, CO
	Work Experience: Mines IT Service Desk Technical Lead and Mines Security Operations Center Analyst
	Extracurriculars: Reading, web development, game development, mobile development, and more
	**Name:** Anya Streit
	Academic Standing: Incoming third-year undergraduate
	Fields of Study: Computer Science (data science track)
	Hometown: Naperville, IL
	Work Experience: Teacher's Assistant at Mines, Alcon Brake Kits
	Extracurriculars: Reading, writing, local music, photography
	**Name:** Aragorn Wang
	Academic Standing: Incoming third-year undergraduate
	Fields of Study: Computer Science (general track) + Applied Mathematics and Statistics
	Hometown: Littleton, CO
	Work Experience: Lockheed Martin Space, Raytheon, Northrop Grumman. ARIA Labs.
	Extracurriculars: Sci-fi and art history reading, weightlifting, cooking, raves and concerts

References

[1] Nvidia *et al.*, “Cosmos-Transfer1: Conditional World Generation with Adaptive Multimodal Control”, *arXiv [cs.CV]*. Mar. 2025, doi: 10.48550/arXiv.2503.14492.

[2] Nvidia. “Cosmos.” Github. https://github.com/NVIDIA/Cosmos (accessed June 8 2025).

[3] Nvidia. “Isaac Sim.” NVIDIA Developer. https://developer.nvidia.com/isaac/sim (accessed June 8, 2025).

[4] Ultralytics. “Ultralytics YOLO11.” Ultralytics YOLO Docs. https://docs.ultralytics.com/models/yolo11/ (accessed June 8 2025).

[5] A. Kuznetsova et al., “The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale.” (2020). IJCV. doi: 10.48550/arXiv.1811.00982 (accessed June 8 2025).

[6] A. Gupta, P. Dollár, and R. Girshick, “LVIS: A Dataset for Large Vocabulary Instance Segmentation.” (2019). CoRR. doi: 10.48550/arXiv.1908.03195 (accessed June 8 2025).

Appendix A – Key Terms

Term	Definition
ACM	Association for Computing Machinery
API	Application programming interface
AWS	Amazon Web Services, used for cloud computing
CUDA	Compute Unified Device Architecture, parallel computing platform developed by NVIDIA.
CSCI	Computer science (Colorado School of Mines department course code)
Docker	Platform that allows for OS-level virtualization
FPS	Frames per second
Git-LFS	Git-Large File Service
HDRI	High dynamic range image
IEEE	Institute of Electrical and Electronics Engineers
IoU	Intersection over Union: measures overlap between predicted and ground truth bounding boxes, used to evaluate model performance
ISO	International Organization for Standardization
LVIS	Large Vocabulary Instance Segmentation
ML	Machine learning
RGB-D	image format that contains both color (RGB) and depth (D) information
ROS 2	Robot Operating System 2, software libraries/tools for developing robots applications
RTX	Ray Tracing Texture Extreme
YOLO	You Only Look Once