Vision-Based GenAI for Robotic Video Captioning (Autonex Initiative)

1. High-Level System Architecture

Designing a multi-camera vision GenAI system for robotic arms involves a pipeline of specialized components. Figure 1 outlines a possible architecture:

- Multi-Camera Input & Sync: Multiple video feeds (e.g. different angles of the robot arm and workspace) are ingested. Since processing is offline, frames can be synchronized and calibrated. Calibration (intrinsic/extrinsic) allows mapping between camera views for spatial consistency. Optionally, a bird's-eye view representation of the scene can be derived to aid spatial reasoning (common in autonomous driving systems
-
- Object Detection & Recognition: Each frame (from each camera) passes through an object detection module to identify and localize relevant items (tools, parts, etc.) and the robot's gripper. Modern detectors like YOLOv8 or DETR can be used, pre-trained on general objects and fine-tuned to the specific industrial objects (even small components). In an assembly inspection example, YOLOv8 was fine-tuned on multi-view images of screws and small parts, yielding high precision

- . The detector should also classify attributes like object color (for example, detect "blue pen") and possibly segment shapes for accurate spatial extent.
- Multi-View Data Fusion: A fusion module combines perceptions from all cameras into a unified understanding. This can be done geometrically (project detections into a common 3D coordinate frame or plane) or via learned features. Multi-camera systems mitigate occlusions and ambiguity – combining views greatly improved detection of small, obscured parts in one study
- . For example, features from each camera's CNN backbone could be merged via a transformer that learns a global scene representation. In calibrated setups, triangulating object positions provides true spatial locations (e.g. "leftmost on table" in an absolute sense, not just per camera).
- Object Tracking & Motion Analysis: A tracking module links detections over time (and across views) to form trajectories for each object and the robot arm. This provides temporal context – e.g. knowing an object moved from point A to B, or that the robot's gripper and an object's trajectories converged. Techniques like ByteTrack (for multi-object tracking) have been used in industrial settings to monitor assembly steps
- arxiv.org

- . By tracking, the system can infer actions: e.g. if an object's bounding box disappears from the table and appears in the gripper, that indicates a "pick-up" action; if it reappears elsewhere, that's a "place" action. Simple spatial logic on tracks (object lifted from surface, moved left, etc.) can yield key facts for captioning.
- Action Recognition Module: In addition to geometric analysis, an action recognition model can classify what the robot is doing (grasping, rotating, assembling) from the video. This could be a 3D ConvNet or Transformer that looks at short video clips. For instance, a model might recognize the pattern of arm motion as a "pick" vs. "push" action. This module provides semantic labels of the type of action, complementing the purely geometric inference. It can be trained on curated examples of each robotic action (possibly using transfer learning from human action datasets, with caution to domain differences).
- Spatial Reasoning: A dedicated component can derive relationships like relative positions ("leftmost part of the table", "in the center of the tray"). This may use the fused 3D positions or a top-down view. For example, after fusing multi-camera data, the system knows all object coordinates on the table and can determine which is left-most or right-most. It may also identify reference landmarks (edges of table, bins, etc.) to express where an action occurred ("from the left bin", "to the top shelf").
- Caption Generation: Finally, a captioning module generates a sentence describing the scene and action. This is a video-to-text model that takes in the processed information (visual features and/or a structured representation of objects & actions) and outputs a natural language caption. The caption should be descriptive and factual (e.g. "The robot picked up a blue pen from the leftmost part of the table."). To achieve this, an encoder-decoder architecture can be used: the encoder ingests a sequence of video frames (or high-level features like detected objects and their trajectories), and a decoder (often a language model) produces the sentence. Classic approaches use a CNN+RNN (LSTM) encoder-decoder, but more recent systems use Transformers for both vision and language
- arxiv.org
- arxiv.org
- . The generation should avoid guessing intent – only visible actions and attributes are described.

Pipeline Summary: Each camera feed is processed for detections; their outputs are fused into a global view where objects are identified, colored, and tracked through time. An action recognition or rule-based logic identifies the manipulation taking place. All this culminates in a caption generator that translates the visual events into a sentence. This modular design ensures the caption is grounded in object recognition, spatial context, and observed motion rather than high-level intent.

2. Pre-Trained Models to Leverage and Adapt

Building this system from scratch is challenging, so leveraging pre-trained or foundational models for each component is wise. These models can be fine-tuned on the new data (once you create a dataset of your specific robotic scenarios). Below is a selection of model candidates for each sub-task:

Component	Model (Pre-trained)	Notes & Relevance
Object Detection	YOLOv8 (Ultralytics) DETR (Facebook/Meta) EfficientDet (Google)	- YOLOv8: State-of-the-art real-time detector, pre-trained on COCO, known to work well on smaller objects when fine-tuned. It has been used in industrial multi-view inspections arxiv.org. Easy to deploy and modify (open-source). - DETR: Transformer-based detector arxiv.org that excels at detecting objects without needing manual anchor tuning. Good for offline use (computationally heavier). Particularly robust in complex scenes and can be trained to detect custom industrial objects. - EfficientDet: A family of detectors with efficient backbones (MobileNet/ResNet) and feature fusion (BiFPN). Pre-trained on COCO, these often perform well on small objects due to multi-scale feature fusion.

Action Recognition	SlowFast (ResNet50/101 backbone) TimeSformer (Vision Transformer) VideoMAE (Masked Autoencoder)	- SlowFast: A two-stream CNN model (one branch captures fast motions, one slow context) pre-trained on large video datasets (e.g. Kinetics-400). It's effective for human actions and can be adapted to robotic motions. Facebook's PySlowFast library provides pre-trained weights. - TimeSformer: A pure Transformer-based video model that treats video frames as a sequence (with spatial patches) and applies attention over time github.com . Pre-trained on Kinetics, it can capture temporal structure of actions. This was used as the encoder in the SpaceTimeGPT video captioning model for its strong performance on action understanding github.com . - VideoMAE: A foundational transformer model trained with masked autoencoding on videos (analogous to BERT for video). It learns general spatiotemporal features without labels and achieves excellent fine-tuning results on action recognition. Using VideoMAE pre-training can help if your dataset is small, by initializing the model with temporal commonsense.

Vision-Language Captioning	SpaceTimeGPT (TimeSformer + GPT-2) BLIP-2 (Vision-Language model) Qwen-Video-7B (Transformer LM)	- SpaceTimeGPT github.com github.com : An open-source video captioning model that pairs a TimeSformer encoder with a GPT-2 decoder. It was pretrained on the large VATEX video-caption dataset and can perform spatial/temporal reasoning in descriptions. Given ~8 frames from a video, it generates a succinct sentence of what happened. This architecture (visual transformer + language model) would be a strong starting point for fine-tuning on your robotic videos. - BLIP-2: A two-stage vision-language model from Salesforce. BLIP-2 uses a pre-trained image encoder (like ViT or CLIP ViT) and a LM (like FLAN-T5 or OPT) with a Q-former to connect them. While BLIP-2 is for images, variants have been adapted to video by processing multiple frames (e.g., the Ego4D video BLIP model huggingface.co). Fine-tuning BLIP-2 on video frames (treating frames as separate image tokens or frames as a batch) could yield a captioner that benefits from large-scale pre-training. - Qwen-Video-7B: A recent 7B-parameter vision-language model (by Tencent) that extends a large language model with video input capability huggingface.co

		. It can take a sequence of frame features and generate text. Such a large model can produce very fluent captions and understand complex events, though fine-tuning it requires more resources. For *simple descriptive* captions, a smaller model (like GPT-2 or T5 based) might suffice and be easier to train from scratch.

Model Cards & Resources: Many of these models have publicly available weights and code. For example, the SpaceTimeGPT repository provides training code and a HuggingFace checkpoint for video description

github.com

github.com

. Likewise, YOLOv8 is available through the Ultralytics GitHub and can be trained on custom data. Pre-trained transformers (TimeSformer, ViT, GPT-2, etc.) are accessible via libraries like Hugging Face 🤗, which also offers vision-text models (see *video-captioning* model hub

huggingface.co

). Leveraging these ensures you start with strong base knowledge (e.g. common object features, basic language) and then adapt to the specifics of your robotic tasks.

3. Multi-Camera Fusion Strategies

Handling multiple cameras is crucial for an accurate and coherent understanding of the scene. There are a few strategies to fuse multi-camera inputs:

- Early Fusion (Image-level): Combine images from different cameras into one representation before feeding to the model. For example, if the cameras cover different parts of the scene, one could stitch them into a panorama or a mosaic (after geometric calibration). However, for separate viewpoints (e.g. front and side view of a table), simple stitching is not feasible. Instead, one might project each camera's view onto a common Bird's-Eye View (BEV) plane (if the ground plane/table plane is known). This is inspired by autonomous driving: models like BEVFormer use transformers to project multi-camera features into a unified top-down map
- ecva.net
- . A BEV can represent object locations in a camera-agnostic way, making spatial relations explicit. Early fusion with a learned model could mean feeding all camera images at once into a single network (e.g. as multiple image tokens in a transformer).

- Late Fusion (Decision-level): Process each camera separately and then merge results. For instance, run object detection on each view independently, then merge the detected object lists (perhaps by matching the same object seen in multiple views). The system can then reason on the combined set of observations. This approach can use simpler geometry: e.g., if camera calibrations are known, an object's 3D position can be triangulated from two views, resolving which detections correspond to the same physical object. Late fusion is robust and interpretable – many industrial solutions first get per-camera detections then perform cross-view matching
- arxiv.org
- arxiv.org
- .
- Feature Fusion (Mid-level): A powerful approach is to fuse at the feature level using neural networks. Here, each camera's video frames go through a backbone (CNN or transformer) to produce feature maps or embeddings. These features are then combined via a fusion network – for example, concatenating and passing through another layer, or using cross-attention. A transformer-based fusion can allow each view to attend to others, learning to emphasize the camera that best observes a given object or action at each time. Research in multi-view action recognition has explored such attention mechanisms to combine views adaptively
- pmc.ncbi.nlm.nih.gov
- . You can also use a hypergraph or graph neural network to represent each camera's observation and fuse them (as seen in some multi-view activity recognition frameworks).
- Spatial Alignment: No matter the fusion method, having the cameras spatially aligned improves coherence. You may perform one-time calibration to get a common coordinate frame. Then the system can maintain a map of object positions (e.g., all objects on the table with x,y coordinates). When the robot moves an object, that movement can be traced in the map regardless of which camera sees it. This enables captions like "from the leftmost part of the table" – the leftmost position is defined in the real world, not just a single camera view. If calibration is available, the fusion can promote 3D understanding of the scene (even if just on a planar surface). If calibration is not available, an alternative is to designate one camera as primary for spatial descriptions and use others only to fill in what the primary might miss (occlusions).
- Temporal Fusion: Since it's video, fusing over time is also important. Multi-camera fusion should consider sequences (e.g., track an object as it leaves one camera's field of view and enters another's). Approaches like the NVIDIA multi-camera tracking pipeline use object re-identification features to keep identities consistent across cameras
- developer.nvidia.com

- . In BEVFormer, not only were multi-camera images fused, but temporal cues were integrated to improve detection of occluded objects and motion estimation
- ecva.net
- . In our captioning system, ensuring that the multi-view fusion is aware of the *temporal continuity* (via tracking or via a temporal model) will make the described actions more accurate (no sudden “object vanished” if it’s actually just moved to another camera).

Key takeaway: Multi-camera fusion will improve the system’s robustness. It resolves occlusion issues and yields better spatial awareness – indeed, using multiple viewpoints was shown to significantly improve perception accuracy in both research and industry

arxiv.org

ecva.net

. The system should be designed to aggregate information across views either through learned attention (for complex reasoning) or through geometric and logic rules (for reliability and interpretability), or a combination of both.

4. Dataset Creation and Training Pipeline

Since no ready dataset exists for “robot arm videos with captions,” you will need to build a dataset from scratch. This is a non-trivial but manageable task with the right tools and strategy:

- Simulation & Synthetic Data: One efficient way to get started is using simulation to generate labeled data. Tools like NVIDIA Isaac Sim, Unity (with the Perception toolkit), or Blender can simulate a robotic arm in a virtual environment performing tasks. Synthetic data can provide perfect labels: you know the object identities, positions, and actions at each time step. For example, you could script a Unity scene of a robot arm picking and placing colored objects on a table; the engine can output the ground-truth bounding boxes, segmentation masks, and even a description of the action (since you scripted it). Domain randomization (random lighting, textures, slight shape variations) will help the model learned from synthetic data generalize to the real world. Synthetic video data can rapidly cover many variations (different object colors, positions, backgrounds) before you collect any real videos.
- Manual Annotation Tools: For real video data (or rendered video), you’ll likely need to annotate frames and write captions. Modern annotation tools can significantly ease this. CVAT (Computer Vision Annotation Tool) is a popular open-source platform for image *and video* labeling
- github.com
- . It provides an online interface to draw bounding boxes, track them through frames, and export annotations. CVAT (and others like LabelStudio or VGG VIA) allow adding custom labels – you could label object classes, colors, etc., and also have a category for “action” on a video segment. For caption

annotation, you might divide a video into clips each containing one atomic task and write a sentence for each clip.

- Semi-Automatic Labeling: To reduce the burden, consider a pipeline where detection/tracking is first done automatically, then a human annotator verifies or corrects it. For instance, you could run a pre-trained YOLO on your videos to get initial object boxes, then use CVAT to refine those rather than drawing from scratch. Research has shown that semi-automatic ground truth generation can improve efficiency and even model accuracy, as consistent labels are obtained
- arxiv.org
- . You can also use the robot's internal data if available – e.g., the robot arm's joint logs could tell you when it is grasping – to automatically label action segments in time.
- Synthetic-to-Real Transfer: If using synthetic data to train initially, you may employ transfer learning techniques. Pre-train your caption model on a large synthetic dataset (even with auto-generated captions using templates like "robot picked up [obj] from [loc]"). Then fine-tune on a smaller set of real annotated videos. Simulation environments like RLBench (a large collection of simulated robot tasks) or AI2-THOR/iGibson (for indoor activities) might provide a basis to generate such scripted demonstrations. Even if their native "captions" are just task names, you can expand them into descriptive sentences as annotations.
- Data Coverage: Ensure the dataset covers the variability you expect: different object types (pens, tools, parts), colors, positions (left, right, middle of workspace), and action types (pick, place, push, etc.). Each video clip in the training set might come with metadata (bounding boxes per frame, plus a ground-truth caption). An alternative is a two-stage training: first train the vision modules (detector, action classifier) on labeled data (with perhaps thousands of frames labeled for detection), and separately train a captioning model on higher-level descriptions (with maybe only hundreds of videos labeled with sentences). The caption model could take as input not raw pixels but a symbolic intermediate representation from the detectors (for example a list of detected objects and their motion attributes). In that case, your caption training data could even be synthesized by programmatically converting the sensor data into sentences (template-based), to get a basic caption model that you then refine manually.
- Labeling Relative Phrases: To enable captions with spatial phrases like "leftmost part," you may incorporate that in annotation instructions. One approach is to label object coordinates and later compute such relations automatically (e.g. a script that knows which object was leftmost and inserts that phrase in the caption). This way, annotators don't have to subjectively decide phrasing – it can be consistent. You could divide the caption into fields: *{action} {object} {source_location} {destination_location}*. For example, *{picked up} {blue pen} {from the leftmost part of the table}*. Maintaining a structured

annotation (perhaps in a spreadsheet or JSON) will help generate training captions and also evaluate the model's outputs.

- Existing Data for Pretraining: Although a custom dataset is inevitable, you might bootstrap with related datasets. For detection, COCO (common objects) or any industrial parts images can help pretrain your detector. For action recognition, human-manipulation video datasets like Epic-Kitchens (egocentric videos of picking up objects in kitchens) or Something-Something V2 (generic object interactions) could provide examples of "pick up cup," "move object" actions – not exactly robot arms, but similar motions with hands and objects. For captioning, large-scale video description datasets like MSR-VTT or VATEX contain generic videos with captions; they are not focused on robotic tasks, but fine-tuning a model like SpaceTimeGPT on your domain-specific data after it has seen thousands of generic videos will likely speed up learning.

Tools Summary: Utilize CVAT or similar for annotation management (many annotators, tasks, tracking)
github.com
. Use simulation tools to generate an initial corpus of labeled videos (possibly thousands of procedurally generated scenes). Leverage scripting to automate as much labeling as possible (object detection models for bounding boxes, known scene geometry for spatial terms). The combination of these techniques will yield a robust training dataset to then train the system from scratch (with heavy transfer learning usage under the hood).

5. Relevant Projects & Research for Jumpstart

To accelerate development, it's worth examining existing projects and literature at the intersection of video understanding, robotics, and captioning:

- Video-to-Command Models: *Yang et al. (2019)* proposed a system to learn robot actions from demonstration videos by generating textual commands
- arxiv.org
- . Their architecture (called GNet + CNet) first uses an object-centric grasp detection network (GNet) to focus on the manipulated object, then a captioning network (CNet) produces a description of the action
- arxiv.org
- . This two-stream idea (global scene + local object features) improved performance on a UR5 robot doing pick-and-place. It's a useful reference for structuring your model: e.g., you might include a module that zooms in around the robot's gripper to capture the object being grasped, feeding those features to the caption generator so it mentions that object specifically.
- Egocentric & Exocentric View Captioning: *Kang & Han (2023)* introduced the GAI dataset (Global-Action-Interaction) for describing robot egocentric videos with external context

- link.springer.com
- . They found that providing both a first-person view (robot's camera) and a third-person view leads to richer captions for human-robot interaction scenarios. In their experiments, descriptions combining global context, the robot's action, and the interaction yielded the best results
- link.springer.com
- . This supports the idea of multi-view input: by training on multi-perspective data, their models generated better explanations of what the robot is doing. While their focus was HRI (explaining robot behavior to humans), the concept applies to any multi-camera setup – consider reading their paper for insights on how they structured descriptions and models for two simultaneous video feeds.
- Spatially-Aware Video Captioning: Some research explicitly targets spatial description. For example, *Li et al. (2021)* in "Robotic Indoor Scene Captioning from Streaming Video" (ICRA 2021) address generating informative captions of a robot's environment. They emphasize important objects and spatial relations in indoor scenes. Additionally, an approach called dense video captioning attempts to describe multiple events in a long video, which might be useful if your system needs to generate a running commentary for a sequence of tasks rather than one caption per video. While dense captioning is more complex, techniques from it (like event segmentation and region attention) could be adaptable here.
- Open-Source Projects and Libraries: Leverage existing frameworks whenever possible:
    - Detectron2 / MMDetection for detection: These provide training pipelines for detectors (including Faster R-CNN, DETR, etc.) which you can adapt to your objects. They come with augmentation, multi-GPU training, and are well-tested.
    - MMAction2 (OpenMMLab) for action recognition: an open-source toolbox that has implementations of popular action models (SlowFast, TSN, TSM, etc.). It can help fine-tune a model on your custom video clips of actions with minimal coding.
    - Hugging Face Transformers for vision-language: With libraries like 🤗 Transformers, you can use models such as ViT, Swin, or TimeSformer encoders and pair them with language models (they even have VisionEncoderDecoder classes). The SpaceTimeGPT model mentioned above is available on Hugging Face Hub
    - github.com
    - , which means you can load it and try generating captions on your own videos, then fine-tune it on new data.
    - Describe Anything (DAM) – an ICCV 2025 project by NVIDIA
    - describe-anything.github.io
    - describe-anything.github.io

    - – is a cutting-edge model that generates richly detailed captions for specified regions in images/videos. While DAM's level of detail ("thick, reddish-brown coat and a bushy tail…" for a dog) exceeds what you need, its architecture (a multimodal transformer with region prompts) is relevant for *localized* descriptions. It shows the potential of modern generative models to output spatial detail. For your simpler use-case, a smaller model fine-tuned to output single-sentence descriptions of the robot's action is likely sufficient, but knowing these advanced models exist means you could incorporate ideas like region-based prompting if needed (e.g., focusing specifically on the robot's end-effector region for captioning that region's activity).
- Industrial and IIoT context: Since Autonex is about AI+IIoT in manufacturing, also explore any industry-focused solutions or research. The multi-camera assembly inspection work by *Nazeri et al. (2023)*
- arxiv.org
- arxiv.org
- (cited above) is one such example where they built a system for quality control. While they did not generate captions, they demonstrated multi-camera vision improving detection of assembly errors. Integrating such a system with a captioning model could yield an automated reporting tool ("Screw on Panel A is missing" – a form of captioning an anomaly). Keep an eye on open-source IIoT projects on GitHub – some projects combine ROS (Robot Operating System) with AI vision for tasks like pick-and-place, which might have code for multi-camera calibration or object recognition that you can reuse.

In summary, by composing a high-level architecture with dedicated modules and leveraging pre-trained models, you can efficiently build a system that observes a robotic arm from multiple angles and narrates its actions. The fusion of multi-view visual data, combined with object-level and motion-level understanding, will enable captions that are accurate and grounded in the scene (e.g. identifying the correct object and its location). A well-planned dataset (potentially bootstrapped with synthetic data and refined with human annotations) will be key to training this system from scratch. With the above models, tools, and references, the development can be jump-started – aligning well with Autonex's goals of integrating AI into physical industrial processes, and ultimately providing clear, real-time (or offline batch) descriptions of robotic operations for monitoring or documentation purposes.