SAND2024-01605J

Machine learning at the edge to improve in-field safeguards inspections

Nathan Shoman[a,*], Kyle Williams [a,1], Burzin Balsara [c,1], Adithya Ramakrishnan [b,1], Zahi Kakish [a,1], Jamie Coram[a], Philip Honnold[a], Tania Rivas[a], Heidi Smartt[a]

[a]*Sandia National Laboratories, 1515 Eubank SE, Albuquerque, 87123, United States*
[b]*University of Michigan, 500 S State St, Ann Arbor, 48109, United States*
[c]*Stanford University, 450 Jane Stanford Way, Stanford, 94305, United States*

Abstract

Artificial intelligence (AI) and machine learning (ML) are near-ubiquitous in day-to-day life; from cars with automated driver-assistance, recommender systems, generative content platforms, and large language chatbots. Implementing AI as a tool for international safeguards could significantly decrease the burden on safeguards inspectors and nuclear facility operators. The use of AI would allow inspectors to complete their in-field activities quicker, while identifying patterns and anomalies and freeing inspectors to focus on the uniquely human component of inspections. Sandia National Laboratories has spent the past two and a half years developing on-device machine learning to develop both a digital and robotic assistant. This combined platform, which we term INSPECTA, has numerous on-device machine learning capabilities that have been demonstrated at the laboratory scale. This work describes early successes implementing AI/ML capabilities to reduce the burden of tedious inspector tasks such as seal examination, information recall, note taking, and more.

Keywords: Nonproliferation, International Nuclear Safeguards, Machine Learning, INSPECTA

1. Introduction

The mission of international safeguards is "... to deter the spread of nuclear weapons by the early detection of the misuse of nuclear material or technology. This provides credible assurances that states are honouring their legal obligations that nuclear material is being used only for peaceful purposes" [1]. Specific safeguards activities performed to provide these assurances vary based on facility type and from state to state.

Despite these variations, there are many common and repetitive inspection activities performed by inspectors; reviewing facility bookkeeping, physically inspecting and maintaining safeguards equipment, taking measurements and samples, examining and verifying seals, item counting, reviewing surveillance images, verifying design information, and generally observing a facility for discrepancies.

Many inspection activities are mentally and physically repetitive and are consequently prone to human error. This is particularly taxing when taken in the context of an increasing number of nuclear facilities under international safeguards and a push for inspectors to perform more investigative activities in addition to their traditional audit activities.

Artificial Intelligence (AI) and Machine Learning (ML) are quickly becoming a fixture of day-to-day life; from smart home controls to assistive chatbots and automated driver assistance. Integrating these advanced capabilities into the international safeguards inspection process could improve the effectiveness and efficiency of safeguards activities, particularly for tasks that are tedious, challenging, and prone to human error.

Sandia National Laboratories is developing a prototype for an AI-enabled smart digital assistant (SDA) for safeguards inspectors to support their increasingly challenging task requirements. This SDA, termed INSPECTA (International Nuclear Safeguards Personal Examination and Containment Tracking Assistant) is designed to work alongside humans to improve inspection execution. INSPECTA is designed to function similar to other commercial SDAs, like Apple's Siri or Amazon's Alexa, which aid with common tasks like note-taking, alarms, and timers. Additionally, INSPECTA is designed to have several safeguards-specific capabilities like using optical character recognition (OCR) to read seal numbers or information recall from the safeguards glossary.

The INSPECTA platform extends beyond capabilities described in this document. For example, advances in robotics pathfinding and point cloud mapping have been made to help improve a variety of inspection tasks. This document narrowly focuses on machine learning technologies that are applicable for nonproliferation tasks so other supporting capabilities are not discussed in detail.

*Corresponding author
Email address: nshoman@sandia.gov (Nathan Shoman)
[1]Equal contribution

2. Design philosophy

The INSPECTA platform consists of two components: an on-device application to be operated by a human and a robotic component that supports automation of some safeguards tasks. These two components are designed to have some similar capabilities, but each platform has unique offerings. For example, for the seal examination task, a human inspector could use the on-device INSPECTA application to automatically scan and record seal identification codes. Alternatively, a robotic platform running the same algorithms could be tasked to autonomously scan and record seal identification codes without requiring human input. The goal of having two platforms is to enable a reduction in repetitive tasks through semi-autonomous robotics or for more dangerous tasks (e.g., inspections in highly radioactive areas) when possible, while leveraging the on-device component when robotics are unavailable.

Privacy, efficiency, and usability are key components of the INSPECTA platform. Data collected from inspection activities is often safeguards confidential and must be protected through a variety of cybersecurity best practices. Facilities frequently prohibit wireless communication or might otherwise be dead zones due to facility construction which makes reliance on cloud-based capabilities impossible. Consequently, INSPECTA is being developed by exclusively focusing on local capabilities that do not depend on external compute devices.

2.1. On-device application

One of the two core components of the INSPECTA platform is the on-device component. Generally, INSPECTA needs to be deployed on a hand-held device with easy-to-use controls that can be effortlessly carried during an inspection. Key requirements for the on-device component included security, privacy, usability, and ability to leverage existing open source components. An early design decision involved consideration of the use of existing platforms such as Alexa, Siri, or open-source options such as Mycroft, as a baseline to build from. At the time of first consideration, Alexa and Siri had strong cloud-based dependencies which could be problematic for facilities that prohibit wireless connectivity. Privacy is also a concern since data would be transmitted through third party servers. Mycroft was briefly considered, however, there were few relevant turnkey skills and limited documentation. These factors lead to the decision to develop INSPECTA without relying on existing digital assistant platforms and instead develop custom capabilities with an inspection use case in mind.

2.1.1. Hardware platform

Hardware used for the on-device component of INSPECTA requires several capabilities; speakers, a microphone, a camera, and a display. While many devices meet this criteria, the team ultimately selected a Google Pixel 6 smart phone as a development device leading to the on-device component of INSPECTA being developed as an Android application. An Android device was selected due to their relative procurement ease, large marketshare, and documented developer tools.

Smart phones are a good development platform for a digital assistant given the variety of internal sensors, some of which may be useful for future capabilities. Smart phones are small, portable, familiar to many inspectors, and allow for communication through both visual and auditory means. Additionally, even commercial off-the-shelf phones have reasonable computational capabilities with a variety of accessories (e.g., extended battery packs). While some nuclear facilities prohibit wireless connections, there is also precedence for allowing devices with physically grounded antennas.

2.1.2. Code platform

The on-device component of INSPECTA is developed in separate modular components which are then integrated into the larger application (see Figure 1). The goal of this modular architecture is to have a more robust set of capabilities that can be reused throughout the INSPECTA platform and combined with modern software testing practices. Activities (dark blue boxes) implemented in the application (light blue box) are based on the capabilities library (green box) and are integrated into the user interface (blue blocks). As INSPECTA is expanded to encompass more safeguards uses cases, new capabilities are added to the library (orange blocks).

The user interface (UI) is implemented using the Xamarin platform [2]. Xamarin is an open-source cross-platform tool based on .NET that can be used to deploy native device applications for Android, iOS, tvOS, watchOS, macOS, and Windows. While the team is currently developing specifically for an Android device, it is unclear how INSPECTA might be deployed in the future. Xamarin allows for a single UI code base for every platform. For example, changing to an Apple device would be relatively easy using Xamarin as the UI code would remain unchanged.

Machine learning capabilities for INSPECTA require on-device processing for data security, privacy, and potentially to meet facility requirements. However, as INSPECTA is designed with a hardware agnostic platform target in mind (as long as sensor requirements are met), a cross-platform, generalized machine learning framework is required. The Open Neural Network Exchange (ONNX) runtime application program interface (API) [3, 4] is being used to that end as it supports a wide range of programming languages and platforms. This contrasts with popular ML frameworks like PyTorch [5] or TensorFlow [6] which support a smaller set of hardware environments. Generally, machine learning models for INSPECTA are developed first in Python using common frameworks (e.g., TensorFlow, PyTorch, or Transformers [7]) and then exported to an ONNX format. Models are also quantized and graph optimized for mobile performance before being added to the main INSPECTA application. In some cases, ONNX is also used to perform data pre and/or post processing.

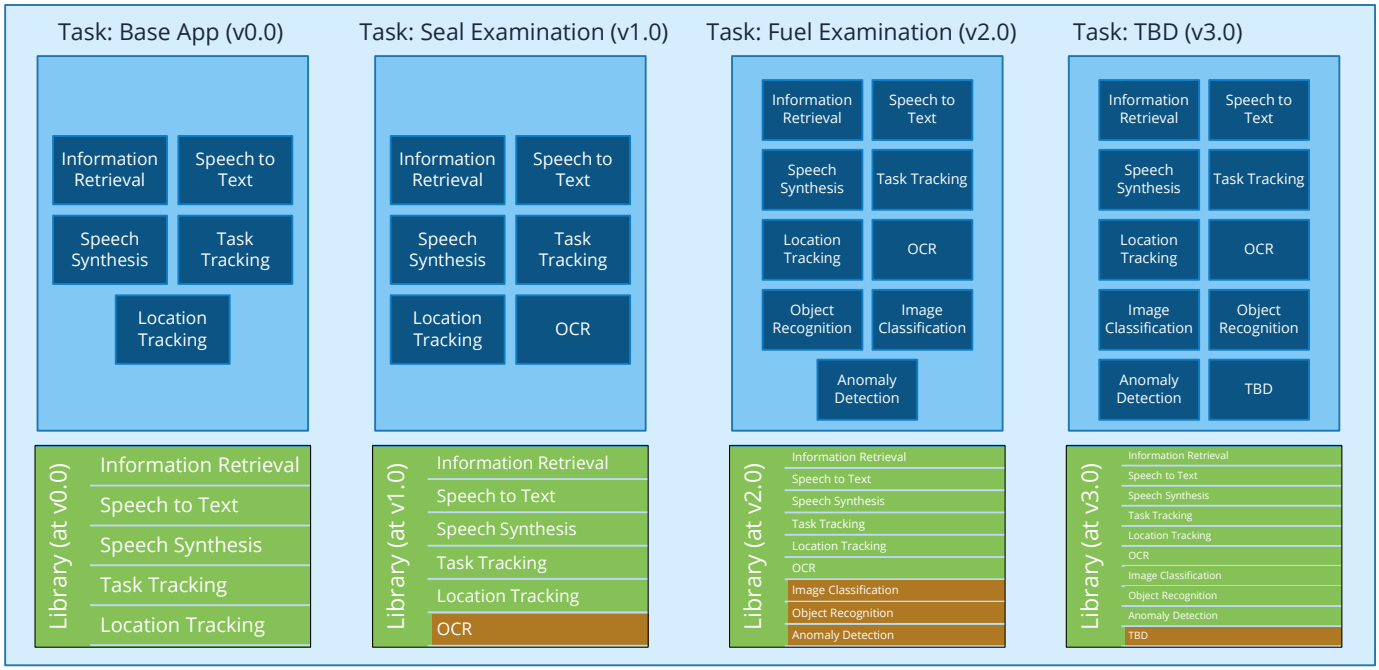

Figure 1: Overview of the on-device INSPECTA application.

2.1.3. User interface

INSPECTA is designed to support both a traditional UI that is associated with phone-based applications and a more language-based interface. While some information is best consumed visually, the team recognizes that inspectors often have bulky gloves and equipment that would inhibit working with a smart phone effectively. The language-based interface incorporates on-device natural language processing for speech recognition and leverages speech synthesis to accomplish tasks without the traditional screen-based UI.

The team leverages a commercial UI kit to build the traditional UI. This allows the team to develop a professional looking application without needing to hand craft elements. The UI was developed in consultation with user interface and user experience (UI/UX) experts. The effectiveness of the UI is demonstrated in the included media wherein the end-user only had about 15 minutes of familiarity with the application before using it for a task. An example of the phone-based UI is shown in Figure 2.

2.2. Semi-autonomous robotics

We intend to further extend INSPECTA by interfacing it with an embodied intelligence, (i.e., a physical robotic platform with enhanced computation), capable of intelligently interacting within an environment. We are currently devising an autonomous system that can interact with inspectors via INSPECTA's mobile UI to complete safeguards related tasks. Since facilities contain dynamic elements and are disparate in their layout, there is a need for ML models of different modalities to help the robot navigate and communicate with an inspector.

The robotics thrust of INSPECTA is currently focused on two interactions: object detection by the robot and communication between the inspector and robot. The first requires a vision-based ML framework capable of running inference on a low-power, single-board computer (SBC) with a sufficient processing rate. The latter requires a

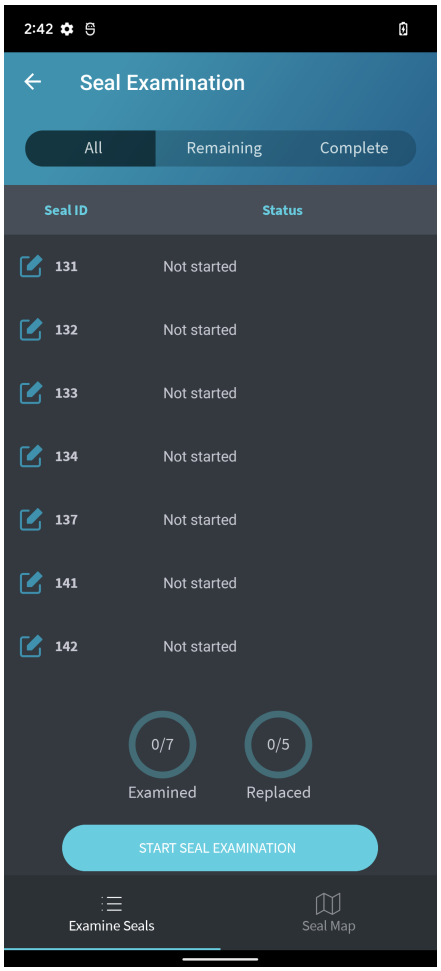

Figure 2: INSPECTA UI showing the seal examination tab.

speech-to-text system that can classify commands spoken to the robot.

3. General capabilities

Currently, the device-based component of INSPECTA is between v1.0 and v2.0 as described in Figure 1. A summary of current capabilities that have been implemented follows:

- Wake word detection (i.e., listens when "Hey INSPECTA" is spoken)
- Automated seal examination (i.e., real-time OCR is used to read seal identification text and compare to a list of known values)
- Information recall (i.e., can extract information given a query and respond with an answer)
- Document ingestion (i.e., perform OCR on documents and prepare them for use with information recall or populate internal databases needed for inspection activities)
- Note taking capability (i.e., notes are typed using an on-screen keyboard or recorded using speech-to-text)
- Variety of convenience functions (i.e., telling the time and reading back notes)

An important guiding principle for on-device machine learning capabilities developed for INSPECTA is computational efficiency. Unlike desktop or cloud-based implementations wherein ample compute resources are available, smart phones and SBCs have limited processing capabilities. Hardware limitations are particularly important in the context of providing a consistent user experience; requiring long wait times for various tasks could lead to end-user frustration and lower platform utilization. Many of the capabilities described in the following sections focus first on identifying a suitable open-source model or algorithm for a given task. Then, the bulk of development effort is focused on optimization of both algorithms and implementation. Most models described below have graph-level optimizations followed by either static or dynamic quantization for better on-device performance. Although INSPECTA is not currently open source software, implementation details are provided in the Appendix.

The robotics capabilities of INSPECTA are based on Spot [8], a quadruped robot developed by Boston Dynamics. Spot is capable of various functionality suitable for inspection [9], logistics [10], and search and rescue [11]. Each leg of the quadruped is comprised of three actuators that drive hip abduction/adduction, hip flexion/extension, and knee flexion/extension [12]. Together, all four legs drive Spot's motion, which resembles the gait of biological quadruped animals.

Spot also contains a perception sensor-suite used for path planning, obstacle avoidance, and localization. There is a stereo camera, a red, green, blue (RGB) camera, and a depth camera on each side of Spot's base; the front of the robot has two of each. The two sensor suites on the front of the robot point in overlapping directions; their information is stitched together to deliver a larger frontal field of view.

Additionally, Spot is equipped with a front-mounted 6-DoF (degree of freedom) articulated robotic arm with a single-joint gripper end-effector. The weight of the arm and any object carried in the gripper is compensated by the orientation and motion of the robotic base. The end-effector houses a 4K RGB camera that can zoom up to 30x for tasks that require more precise image capturing than those taken from the lower-resolution camera sensor suite on Spot's base.

Finally, while users cannot directly manipulate Spot's individual actuators, Boston Dynamics provides a robust Software Development Kit (SDK) [13] written in Python. The SDK abstracts motion, manipulation, and sensing tasks for incorporation into user's preferred client-side applications. Any user-created applications that drive Spot must run on an external computing device and not Spot's

internal computer, which is restricted by the manufacturer. Figure 3 shows a 3D printed payload mount for an Nvidia Jetson [14] SBC, which has been added to Spot to run a variety of algorithms. The SBC connects via ethernet to a port located in Spot's rear. A Wi-Fi chip and antennas added to the SBC allow an inspector to interface with Spot through the INSPECTA app as well.

Figure 3: Spot with mounted Nvidia Jetson companion computer and GPU, complete with wireless connectivity.

In summary, Spot's locomotion, perception and manipulation systems enable it to operate in the same built environments in which inspectors work. Spot can walk on many types of terrain, up stairs, over small obstacles and around large ones. Using its built in arm and gripper, Spot can even open doors. Furthermore, the ethernet interface and SDK allows the team to develop arbitrary autonomous inspection behaviors and deploy them on Spot, while also enabling connectivity between Spot and the INSPECTA app.

3.1. Device: Wake word detection

Wake word detection, or voice activity detection (VAD), is the detection of speech in an audio clip. Accurate detection of speech is important for reducing computational overhead of the on-device INSPECTA application. While INSPECTA has a speech recognition algorithm, which is detailed more in the following section, constantly running the model to look for wake words would not only slow down the device, but also reduce battery life. Instead, it is preferable to run the speech recognition model only for cases where speech is likely present. The three key steps for this task include audio collection, voice activity detection, and speech recognition.

Audio collection: Audio collection for the wake word task is implemented through Android-specific libraries. An audio buffer constantly collects audio and is passed through the voice activity detection module. This process starts at application launch and is only paused if another machine learning task is launched (e.g., speech recognition or another task). The buffer is never written to storage and resides only within volatile memory. While active, audio collection is continuously collected in the background.

Voice activity detection: The original implementation for voice activation detection (VAD) relied on engineered metrics that compared the buffer's statistical properties to a variety of thresholds. However, that approach was brittle and did not work in a wide variety of settings. That approach has been replaced with the Silero VAD model [15], which is small and easy to compute. The audio buffer is passed to the Silero VAD model at a regular interval to determine the estimated probability that speech is present. If a threshold probability is passed, then the speech recognition module is activated.

Speech recognition: Speech recognition is performed if a threshold score from the Silero VAD model is exceeded. The model used here for speech recognition, Meta's wav2vec2.0 [16], is the same as for other speech recognition tasks for the on-device INSPECTA application. Further details of this model are described Section 3.2.

3.2. Device: Speech recognition

Speech recognition is the process of converting audio with spoken word to plain text. A device-specific implementation (i.e., Android-specific) is used for the device-based speech recognition task. The current implementation of speech recognition for the device-based INSPECTA application does not rely on any UI interaction and can be triggered by speech prompts only.

Audio collection: Audio recording starts after the wake word ("Hey Inspecta") is triggered. Specifically, an Android audio buffer constantly collects audio in a pulse-code modulation (PCM) format using four byte floats at a rate of 16 kHz. This buffer is written to an internal `.wav` file while audio recording is active. Audio recording ends after two consecutive buffers worth of audio have voice activation scores below a specified threshold; a likely indication that the speaker is finished speaking.

The most important parameter for the audio collection process is the audio buffer size. Since the buffer itself is appended to the recorded audio file, the buffer size can impact the audio file and subsequent downstream tasks. If a buffer is too small, then spoken words can be cut off and broken in the output audio file. If the buffer is too large, read latency increases and the resulting output file grows in size as extraneous zeros are added; this reduces responsiveness. The values used in the current version of the INSPECTA application were determined empirically by considering responsiveness and accuracy of the predicted speech.

Speech recognition: INSPECTA uses the wav2vec2.0 algorithm [16] for converting the raw waveform stored in the internal `.wav` file to an embedded output. This model is selected for a number of reasons; raw waveforms can be passed as input without further preprocessing, the model

has been widely implemented in several frameworks, and the model performs well when trained on even small (1 hour) quantities of data. High performance in low training data contexts are important as INSPECTA will ultimately be used by inspectors of varying nationalities, so fine-tuning on different accents will become important.

During speech recognition, the audio file from the collection step is first read into memory provided it can fit into memory. Otherwise, it is read into chunks and processed iteratively. Next, the waveform is scaled to the range $[-1, 1]$ to improve performance. The waveform is then passed through the quantized and optimized wav2vec2.0 ONNX model on-device. The output is tokenized using connectionist temporal classification (CTC), so additional post-processing is required. There were no readily available tokenization libraries for the .NET framework available to perform this task originally, however the team attached a decoder to the model resulting in an end-to-end ONNX speech recognition module. The final decoded output, plain text, is used for down stream tasks such as command matching or note taking.

3.3. Device: Speech synthesis

There are a variety of algorithms for performing speech synthesis (i.e., text-to-speech) including WaveGlow [17], WaveRNN [18], and more recently BigVSAN [19]. However, Android offers a built-in text-to-speech service that runs locally on-device. INSPECTA leverages that Android implementation rather than relying on a machine learning approach given the low effort to implement.

3.4. Device: Command matching

Commercial digital assistants have the ability to match spoken commands to software actions (e.g., turn on the light or lookup the current weather). While some aspects of that process are well documented publicly, others, like command matching, are not. There are two general approaches that can be used for command matching; similarity-based and embedding-based. In the former, a metric of string similarity is used to compare the command to a list of options. The embedding-based approach uses a machine learning model to embed the sentence into a different space wherein sentences with similar sentiment are close together. The current implementation uses a Levenshtein distance [20], a similarity-based technique. Specifically, when Levenshtein is used to measure the distance between two sentences, it is the minimum number of single-character edits (i.e., insertions, deletions, or substitutions) required to change one sentence to the other.

INSPECTA attempts to match the speech-to-text result to a list of available commands by calculating the Levenshtein distance between all possible result-command pairs. If the score is under a threshold, then the best command is executed (e.g., start seal examination task launches the seal examination activity). If the score is over the threshold, then it is assumed no command is a good match and information recall is requested instead.

The Levenshtein distance is straightforward, intuitive and has performed well, but might not represent the most effective strategy to perform command matching. However, using embedding-based approaches might require nuclear-specific vocabularies to capture the correct sentiment of a command or sentence. There are some efforts outside of this work to develop language models with nuclear-specific vocabularies, however, they have not been extensively investigated here. Deployment of INSPECTA to more real-world scenarios might require a more robust command matching metric than the currently implemented Levenshtein distance.

3.5. Device: seal examination

Items at nuclear facilities are randomly sampled during inspections to ensure material is present and accounted for. Inspectors apply seals after an item has been sampled to maintain continuity of knowledge between inspections. If, at the next inspection a seal does not show signs of tamper, the item may not need to be measured again. Seals are used in many places such as containers of nuclear material, equipment cabinets, doors, and gates. The IAEA verifies approximately 15,000 metal cup seals each year. The verification procedure includes locating the seals, verifying the identity of the seal matches records, and looking for visual indications of tamper.

INSPECTA currently supports real-time optical character recognition (OCR) to reduce the burden of seal examination on inspectors. The on-device verification activity can currently scan a metal cup seal and compare the identifier to a known list while item-seal checking capabilities (i.e., ensuring the correct seal is on the correct item) are under development. Capabilities related to tamper detection are also under development. Existing capabilities can be used to ensure scanned seals are present on a predefined list.

Two different user experiences were considered. In the first, a user takes pictures of a seal, triggers the verification pipeline on that image, and receives some feedback. If the results are inadequate due to image conditions (e.g., poor prediction due to lighting or camera angle) then the user would have to take another image. This approach would likely result in a tedious and slow workflow that is even worse than manual verification.

Instead, a real-time implementation is chosen wherein OCR and verification are performed on a real-time preview from the camera. This approach results in a better user experience as the user only has to point the camera while moving or rotating the seal, however this approach requires faster processing. If processing is too slow, then the camera preview appears as a slideshow and results in a frustrating user interface. During the real-time preview, bounding boxes for current text in the analysis window, its translated text, and a softmax probability, are all drawn on each frame to provide user feedback. An example of the seal examination procedure is shown in the attached media.

The current real-time seal examination pipeline consists of several steps as follows:

Image collection: There are several ways to generate a camera preview in the UI. The current implementation leverages the OpenCV library, with .NET bindings [21] updated to the latest version of OpenCV (v4.8.0), to create a preview and modify frames of the preview. This is chosen instead of an Android-native approach, which only supports returning frames in RGB format in some cases (varies by Android version and device) and returns YUV format otherwise. Downstream tasks expect RGB format and converting frames from YUV to RGB is not straightforward, so OpenCV preview functions that return RGB frames are used instead. The downside is that OpenCV camera preview resolution is limited to that of the device display, not the maximum camera resolution. However, this limitation did not negatively impact the seal examination pipeline, which is focused on performance and utilizes lower resolution images.

When a frame from the preview is generated, it is then processed by subsequent operations in the seal examination pipeline. Since these operations take a non-zero amount of time, not all preview frames can be processed. Frames that arrive while a frame is being processed are simply dropped. This effectively results in a frame rate drop from the original preview rate to a final rate that is determined by the computational overhead of the pipeline.

Image preprocessing: The baseline preview provided by OpenCV is about 20 fps regardless of resolution. Since processing speed is a priority, the preview image is rescaled to a resolution of 720x480. Next, a blurring filter is applied to the preview covering all except for the analysis region, which is only 160x120 (see Figure 4 for an example). Finally, the image contrast is adjusted by a factor of 2.5. There are a variety of algorithms that can perform OCR on images. Since processing speed is a top priority, contemporary end-to-end transformer architectures were not used. Instead, a two-stage approach based on the EasyOCR library is used.

Text detection: The first stage is to detect text in a scene (i.e., image). Here, the CRAFT model [22] is used, which is based on VGG [23] and U-Net [24] architectures. The outputs of the CRAFT algorithm are region scores, used to localize individual characters in an image, and affinity scores which are used to group characters into a single word. More specifically, region score is the probability that a given pixel is the center of a character and affinity score is the center probability of the space between adjacent characters. These metrics are not directly useful for downstream tasks, so additional processing is required to generate bounding boxes. This is accomplished by applying a threshold to both scores and finding a minimum bounding box on the binary map of the summed components. Once bounding boxes have been identified, the regions of interest are cropped and passed to the second stage of the pipeline.

Text recognition: The second stage recognizes text in an image. This stage uses EasyOCR's recognition model [25] which is based on the CRNN model [26]. Each image segment that is determined to contain text in the previous stage is preprocessed by again having the contrast adjusted by a factor of 2.5. This model uses an Adaptive 2D Pooling layer, which is not currently supported by ONNX. The first attempt remedied this by exporting a fixed sized model which created the need for padding and/or resizing of each input image patch. This resulted in mixed performance, so the current implementation instead approximates the Adaptive average pooling layers with a global pooling layer, which enables variable sized input in the resulting ONNX model that possesses increased performance. The model is trained with the CTC loss, so the outputs are decoded from a sequence of scores to letters that form a word. The resulting text from each cropped image is passed on to the final stage of the seal examination pipeline.

Identification matching: The final step of seal examination is to compare text in the frame to a known list of seal identification. Here, text is directly compared and only direct matches qualify (e.g., a text identifier of "123" must have an exact match from the recognition algorithm). Text is only compared to possible seal identification if the softmax probability from the text recognition stage is at or above 0.85, which acts as a proxy for confidence. If an image is a match for a listed seal, then visual feedback is provided to the user and the image is stored on-device for later human review if necessary.

The real-time seal examination pipeline requires about 85ms per frame resulting in a final preview frame rate of about 12 frames per second. This framerate is not particularly fast from a visual preview perspective, but is quite fast for on-device inference. This performance is achieved by leveraging offline graph optimization (shape inference and operation fusion), static quantization (QDQ format, MinMax calibration), and Android's Neural Network Application Programming Interface (NNAPI) executor.

3.6. Device: Document scanning

Document scanning, or document OCR, involves taking a picture of an image and transcribing any text in the image. This capability is particularly useful when needing to scan documents that need to be stored on INSPECTA for internal use. Use cases could include scanning a list of seals to be used later in the seal examination task or scanning a document to be queried with the information recall module. Document scanning presents two unique challenges not present in the seal examination task. First, documents might have many words; a single page from a text book could contain several hundred. Resolving the individual words requires a higher resolution image than what is used in the seal examination task. Second, pages might be warped or skewed, particularly if they are in a book. This can result in lower performance for the OCR task and also make it more difficult to order words. Unlike for seal examination, it is important to ensure words

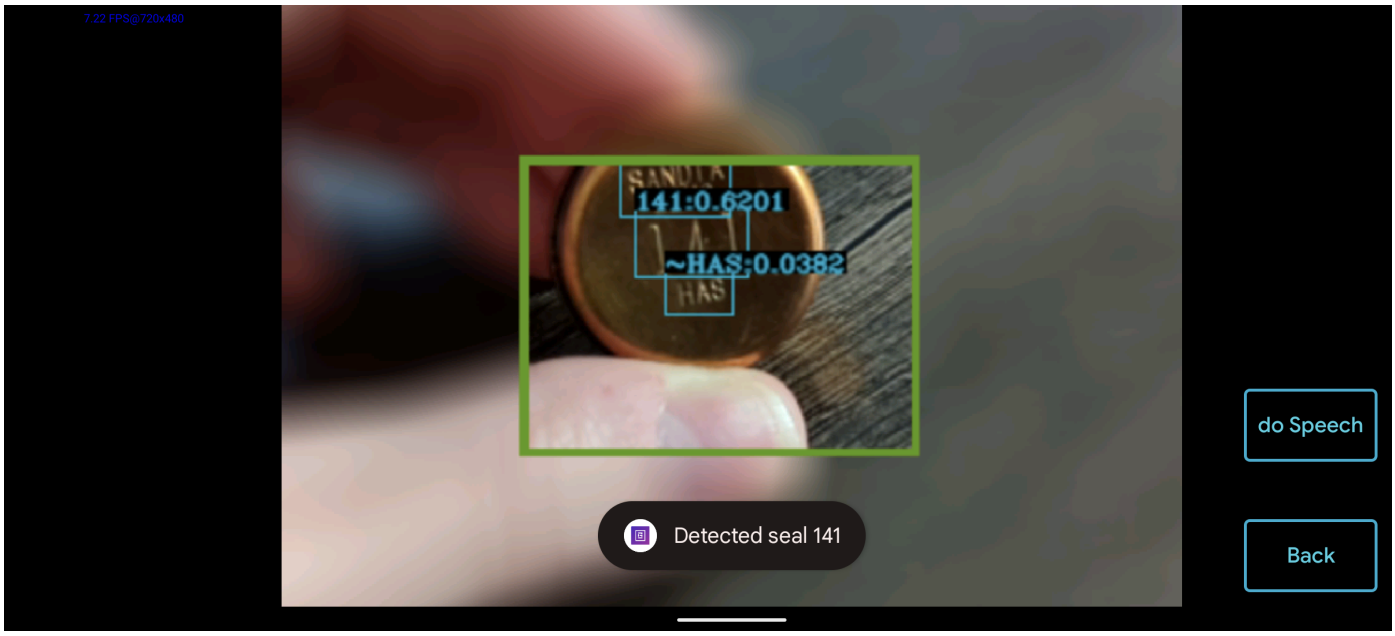

Figure 4: Example of the seal examination task being performed on-device.

in scanned documents are ordered properly from left to right and top to bottom. The OCR pipeline (i.e., detection and recognition) does not necessarily output the bounding boxes in the proper order.

The document scanning task is not performed in real time. Instead, this task focuses on taking a single high resolution image of a document and focusing on accuracy over speed. Attempting real-time predictions, as is done for seal examination, on an entire page of text is simply not feasible using on-device resources. The workflow by the end-user for document scanning involved toggling two switches. The first indicates if an image is free form text (i.e., non-regular text such as that on a sign) or not (i.e., regularly ordered text as on a page). The second controls if the dewarp algorithm, which attempts to correct for skewed pages, is used. The document OCR pipeline can take a considerable amount of time in some cases, so images captured by the end-user are stored on-device. Then, the user can trigger processing of the images manually. This procedure is shown in the attached media demonstrating the user interface. There are several components of the document scanning pipeline.

Image collection: Higher resolution images are required for document scanning, so a different camera backend is used. Instead of using the OpenCV preview backend, the native Android CameraX backend is used to capture images. These images have roughly a 3072p resolution compared to the seal examination's 120p. This is essential for resolving densely arranged text. Image collection is performed when the user presses a button to capture an image. The images are then stored on-device in a queue until the user indicated the queue should be processed. After processing, documents, both image and the predicted text, are viewable in the UI. The predicted text is also available for use with the information recall module to allow interrogation of the scanned document. Several key steps are required for document scanning:

Page dewarping: Dewarping pages that are otherwise skewed or warped improved the OCR pipeline performance and helps to better align translated text. Dewarping text is a challenging problem that sees continual development. Several different approaches were considered for the dewarping problem, both deep learning based (DewarpNet [27] and Displacement Flow Estimation [28]) and traditional image-based approaches (cubic spline model [29]). The two deep learning methods exhibit poor performance on several test cases, so the traditional image-based approach is used. Specifically, a cubic spline model developed by Matt Zucker [29, 30] is used with some modifications to improve performance where possible. There are two key concepts used in this approach. The first is to note that a flat sheet of paper resides in the same z-coordinate plane (i.e., there is height and width, but depth should be the same at all points). If a page is bowed or warped, then the z-coordinates change across the horizontal axis. The change in the z-coordinates can be represented as a cubic spline that depends on the horizontal position as described in Equation 1. It is assumed that the page positions on the left and right side should be fixed, therefore $f(0) = 0$ and $f(1) = 0$ where $f(x) = z$. Representing the curvature by setting $f'(0) = \alpha$ and $f'(1) = \beta$ and solving yields the function expressed in Equation 2.

$$z = ax^3 + bx^2 + cx + d \tag{1}$$

$$z = (\alpha + \beta)x^3 - (2\alpha + \beta)x^2 + \alpha x \tag{2}$$

Next, it can also be noted that there is a world coordinate system with three dimensions, but that a camera can only record two as images do not explicitly capture depth. The transformation between the world coordinate system and camera coordinate system can be expressed by a rotational and translational vector. It is possible to develop an optimization problem expressing the warping of a page when the two models are taken together. First, 2D image points are projected into a 3D space using the cubic spline model. Then, the 3D points are projected back down to 2D space using the camera pose model. An optimization goal can be formulated to minimize the reprojection error thus jointly optimizing the cubic spline model and camera pose model. The models can then be inverted after optimization to find a coordinate translation between the current warped page space to a new, unwarped space. The specific steps are described by Zucker [29] and will not be repeated here.

Several changes were made that resulted in approximately a 30x speedup from the original code. First, the number of spans and points used in the optimization problem were vastly reduced by noting a few lines of text can generally estimate the extent of page warping. A more suitable optimizer is leveraged, namely L-BFGS-B [31] using the Accord.NET library [32]. The code for the optimized page dewarper is available on GitHub (see Appendix 6.5). This process takes about 10 seconds per page when running on-device.

Text detection: Text detection is performed using the same algorithm as the seal examination task; the CRAFT algorithm. The only significant difference is that the input image has a much higher resolution.

Bounding box ordering: Bounding boxes identified by the text detection step are often not in reading order (i.e., left to right and top to bottom). Additional processing is performed to ensure bounding boxes are ordered correctly for the final output text. This is accomplished by converting the image to greyscale and summing across the horizontal dimension. Then, bounds of text lines are determined by looking for significant gaps between the summed rows (e.g., lines with text should be non-zero). After determining the rows, bounding boxes are grouped by row and ordered by horizonal position. The image segments containing individual words are then cropped and sent to text recognition.

Text recognition: Text recognition is performed using the same algorithm as the seal examination task: using EasyOCR's CRNN model. Some preprocessing has changed, particularly the padding around the bounding box that is applied before an image is passed to the text recognition algorithm. Since accuracy is prioritized for the document OCR task, a more contemporary model was also evaluated. Microsoft's TrOCR [33] (small checkpoint) was compared to the EasyOCR CRNN. One particular challenge with implementing TrOCR on-device is that the small version of TrOCR requires a SentencePiece [34] decoder.

SentencePiece is a model specific text tokenizer that requires a model and does not rely strictly on a function call. SentencePiece was not readily available for .NET, the framework used to build the on-device component of INSPECTA. However, ONNX runtime-extensions recently implemented new capabilities, so the team was able to package SentencePiece as a standalone ONNX module. The CRNN model is fast and performs well for well-defined printed scene text, but is sensitive to preprocessing (e.g., noise reduction must vary based on font size and thickness). For that reason, TrOCR is being investigated as a potential replacement provided it is more robust to various text sizes and formats.

Overall, the document OCR pipeline bears similarity to the seal examination pipeline, but emphasises accuracy over speed. The same detection and recognition models are used, but higher resolution input images, and in some cases, page dewarping is required. The CRNN model works well for certain text sizes, but can struggle on freeform text of varying sizes. Current examples of freeform text detection can be seen in Figure 5, whereas the document scanning UI and result is shown in Figure 6.

Figure 5: INSPECTA free-from text identification with translated text.

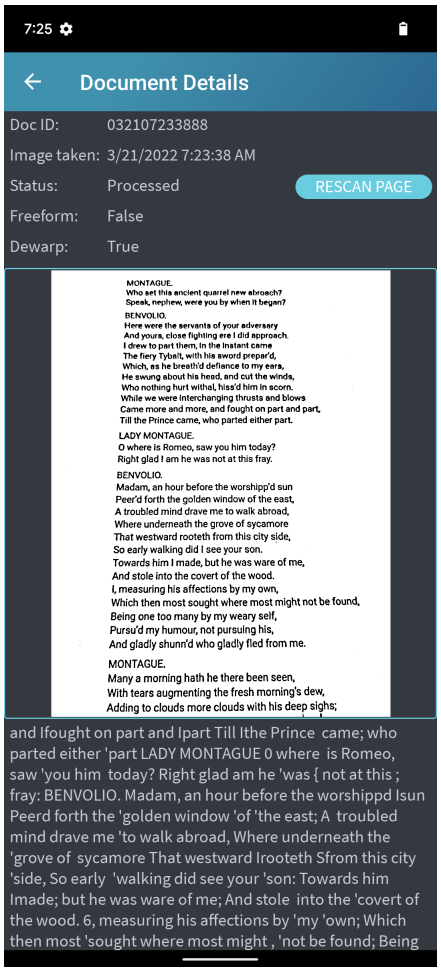

Figure 6: INSPECTA document scanning UI with translated text. Text taken from [35].

3.7. Device: Information recall

The final machine learning module that is currently implemented for the on-device INSPECTA application is information recall. Generally, this refers to the process of asking a natural language question and receiving a natural language answer. Contemporary large language models and chatbots can perform this with ease given the large training data corpus, however, this approach is not available for an on-device environment. The approach used in INSPECTA requires two stages. The user experience for the information recall on INSPECTA involves verbally asking a question followed by an answer that is spoken via the speech synthesis model.

Context generation: Question and answering with models that can fit on-device often requires context that contains the correct answer. Generating this context can be challenging particularly when there might be multiple documents with hundreds of pages. There are both non deep learning approaches (e.g., ElasticSearch [36] or FAISS [37]) and deep learning based approaches. INSPECTA currently utilizes a deep learning approach due to the relative ease of implementation and associated permissive license models.

The context generation step is accomplished by utilizing a DistilBERT [38] model that is fine-tuned on the MSMarco [39] dataset. First, documents are converted to raw text, either through a PDF conversion tool for existing documents, or through the document OCR pipeline for images. Next, the text is split into smaller segments of about 700 characters long. The document segments are then encoded by applying a WordPiece tokenizer [40] and passing them through the DistilBERT model. Relevance of text segments for a particular question are measured by multiplying the embedded query with the embedded text segments. The top-5 scoring segments are decoded and used as context for the next stage.

Embedding of text is performed offline where possible to save computational time. For example, INSPECTA currently has the IAEA safeguards glossary [1] as a reference, which is split, embedded, and stored as an internal file during the application complication stage before deployment to the mobile device.

Question and answering: The second stage combines context and a query to generate an answer. Here, a SciBERT model [41] that is finetuned on the SquadV2 dataset [42] is used. The query and context are both tokenized using WordPiece and concatenated before being passed to the SciBERT model as input. Postprocessing is then required to ensure the query is not returned as the answer. The top scoring answer candidate is decoded and then returned. The on-device INSPECTA provides this answer through the synthetic speech module.

The information recall module generally exhibits mixed levels of performance. Query and answer pairs with more nuclear specific vocabulary tend to perform poorly whereas more general knowledge pairs work well. This likely relates to poor representations of nuclear-specific concepts in the underlying model vocabulary. Two different approaches are being considered to improve performance. First, the context generator could be pretrained on a more nuclear-specific dataset (e.g., OSTI dataset as demonstrated in Burke [43]). Second, the question and answering module could leverage the "salt-and-pepper" technique demonstrated by Wilson [44] to generate a better context for nuclear-specific terms.

3.8. Robotics: Object detection

For the robotic component of INSPECTA, Spot needs object detection to better understand its environment to semi-autonomously perform inspection activities. One way to accomplish this is through implementing object detection algorithms. Object detection is a branch of computer vision that involves classifying and localizing objects of interest in an image or video. This is done by computing a compact bounding box around a detected object and providing a confidence score associated with its classification. Specifically, object detection allows Spot to detect objects of interest during autonomous inspection, e.g. drums storing nuclear material and the presence and location of their

seals, along with other common objects such as people, chairs, tables etc.

The robotic component of INSPECTA uses the YOLO (You Only Look Once) algorithm for this capability, which is an incredibly fast state-of-the-art algorithm that computes bounding boxes and class probabilities all in one shot. This is a faster and more efficient method compared to predecessor algorithms, such as sliding window or region proposal methods, that required multiple computation steps and ran significantly slower. Speed is a high priority when performing object detection on video streams rather than still images. YOLO works by dividing an image into a grid of cells which are all run through a neural network at once, in parallel. The network computes bounding boxes and class predictions for each grid cell, given the context of the entire image [45]. Next, Intersection over Union (IOU) and Non-Max Suppression are used to filter out low-confidence or duplicate predictions, so that each object has a single bounding box and prediction associated with it.

There are many iterations of the YOLO algorithm, YOLOv5 (version 5) [46] is used here. YOLOv1 introduced the major algorithmic ideas, while v2 through v4 were largely incremental optimizations. YOLOv5 is easier to use than its predecessors, is written in Python and Pytorch, and has many built-in tools for data augmentation and training monitoring.. YOLOv5 comes pretrained to classify all the objects in the Common Objects in COntext (COCO) dataset [47].

INSPECTA runs real-time object detection on video streams from Spot's cameras, which are captured through the SDK. All neural computation is perfomed on the Nvidia Jetson mounted on Spot. Pretrained YOLOv5 applied to Spot's six on board cameras is shown in Figure 7. Note the pretrained model makes many erroneous classifications, but these initial results are still promising and can be improved with more data and tuning.

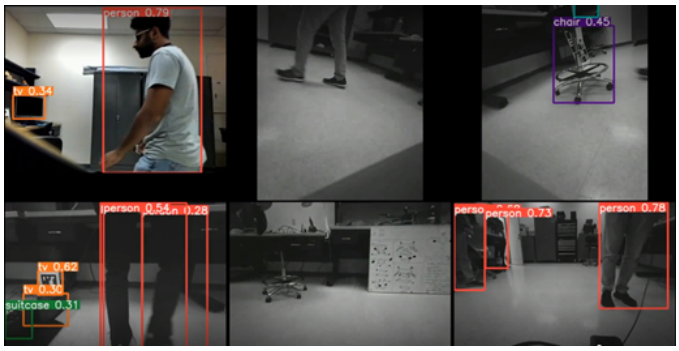

Figure 7: YOLOv5 on Spot.

While common objects are useful, INSPECTA has more specific needs.The team uses transfer learning to fine tune the pre-trained YOLOv5 model with the Limbo dataset [48]. Limbo contains synthetic images of various uranium hexaflouride containers which are part of the nuclear fuel cycle. Early results on training data are shown in Figure 8.

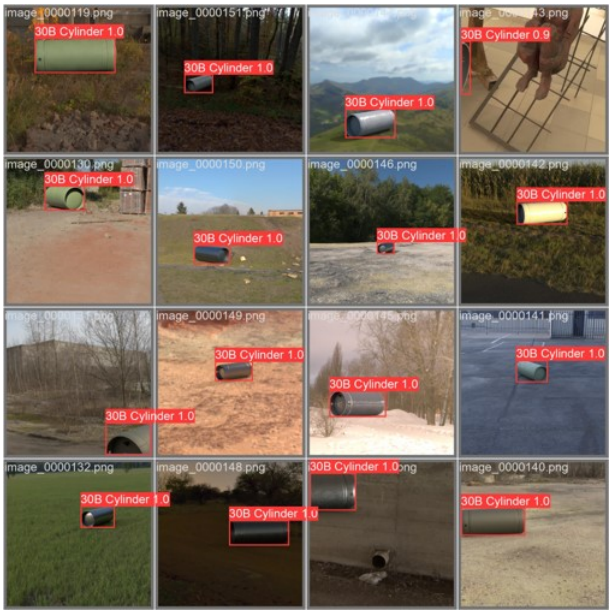

Figure 8: Initial results of YOLOv5 on Limbo dataset training data.

Next, to facilitate the move from synthetic data to a physical laboratory environment, the team created custom in house datasets for identifying drums and seals. These custom datasets are created by capturing videos and extracting images/frames containing drums and seals under different conditions (i.e. varied lighting, size, occlusion, etc). Some of these training images are blurry (due to motion during capture of these videos), which is advantageous since these are representative of the motion blur experienced by Spot when it moves and takes video. These images are then labeled using LabelMe [49] and Roboflow [50], and the pretrained YOLOv5 model is fine tuned for these objects via transfer learning. Figure 9 shows four example images from the training set, with bounding boxes of different colors as a visual representation of the labelling process.

Figure 9: Examples of training images with red bounding boxes for drums and green for seals.

So far, training has been completed on the drum data, but not the seal data. Results of this training are shown on test data in Figure 10.

Figure 10: YOLOv5 on in house drum dataset test data.

3.9. Robotics: Speech recognition

To reduce effort and/or training for an inspector teaming with Spot, a voice control module was added to the robotic INSPECTA platform. It currently enables an operator to control the basic functionality of Spot's movement and is integrated with Spot's underlying behavior tree. We mounted an additional microphone and speaker on Spot to enable this functionality.

Similar to the voice command option in the on-device app, the module implemented here utilizes the pre-trained models in the wav2vec2.0 [16] PyTorch library. However, to enable this part of INSPECTA to work independently from the on-device app, the voice controller is deployed on the Nvidia Jetson which is directly connected to Spot via ethernet. This has the added the benefit of expediting the processing time of Wave2Vec2 using the Jetson's GPU, as compared to the relatively limited computation of the on-device app.

Voice commands are initiated using a push button system connected to the Jetson's general purpose input/output (GPIO) pins, which triggers a recording window. During this time, INSPECTA captures the desired command using a multichannel spatial microphone controlled by the Linux-based recording utility, arecord. Next, the audio is processed by the wav2vec2.0 model, and the output is fed into a simple filter that uses the Levenshtein distance to convert potentially incorrect spelling or misnomers into desired commands. The multichannel microphone carried by Spot also features a speaker, which is used to repeat the command out loud once it is processed. Text-to-Speech is done with the pyttsx3 library. This enables the operator to be aware of any movement prior to its execution and cancel the command if necessary. Spot is shown with the mic/speaker and prototype push button interface installed in Figure 11.

Current Spot motion commands include walking, sidestepping, or rotating in place by a set angular displacement. Additional commands have been added to incorporate Spot's existing behavior tree, which includes a command to locate and move toward a fiduciary marker, as a proxy for visually identified inspection relevant objects. The goal of the behavior tree is to have simple behavioral

Figure 11: Spot with mic/speaker and prototype push button interface installed.

entry points to standard sets of flexible inspection procedures.

Initial attempts at running vocal command using only the CPU on the Jetson resulted in processing times on the order of ten to twenty seconds. After switching to GPU processing, the voice controller now processes commands in under a second. Additionally, the push button system enables the Spot-INSPECTA duo to be completely standalone, without need for any desktop terminal connection, or the on-device app. However, this system is still limited in the need for tactile interaction with Spot, which may be inconvenient in some inspection scenarios. Trigger word detection should be investigated further as a true hands-free solution for the operator. Additionally, while the setup works well in a controlled lab environment, further testing must be done in noisier scenarios. To overcome the difficulty of operating in such environments, there are two proposed solutions: investing in a microphone with better noise mitigation and utilizing audio post-production to filter the waveform before command recognition is run, at the cost of some processing time. Finally, this system worked when tested on audio tracks from the team, but further testing may be required to ensure adequate performance on the variety of accents present in the inspection community.

3.10. Device-Robot Interface

The team has conducted some initial work towards fully integrating the on-device app and robotic components of INSPECTA. The initial prototype is a simple transmission control protocol (TCP) connection that exchanges text and images. A diagram is shown in Figure 12.

4. Discussion and future work

There are a number of ongoing challenges for the capabilities demonstrated here that are not readily solved using off the shelf components. The wake word pipeline,

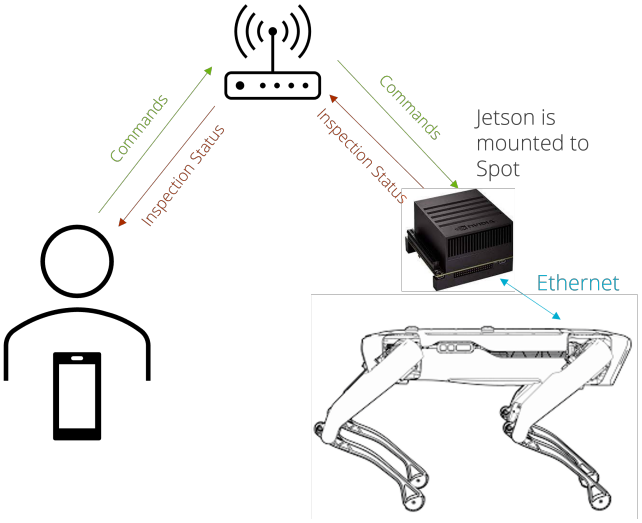

Figure 12: INSPECTA connectivity diagram.

while significantly improved, still exhibits mixed performance particularly for recognizing the wake word. There is no guarantee that the audio buffer contains the full wake word audio, which can only be partially filled depending on when the buffer is read. Comparison of the translated speech to a list of hard coded possible permutations of "Hey Inspecta" using Levenshetein distance can also lead to missed activations. One potential solution would be to replace the VAD and Speech Recognition pipeline with an end-to-end machine learning wake word approach (e.g., OpenWakeWord, Picovoice Porcupine, and Fluent Wakeword).

OCR works well for the seal identification task, but challenges remain for document scanning. First, the preprocessing stages in the dewarping step have a strong impact on downstream recognition tasks. While two different deep learning methods for page dewarping were considered, performance was found to be inferior to the existing cubic model approach. Document scanning might be improved by either using a different optimization algorithm or by fine-tuning a deep learning approach. The text recognition portion of the document scanning pipeline could also be improved through better optimized preprocessing steps. Further, TrOCR is being considered to improve the recognition portion of the OCR pipeline as it represents the current state-of-the-art in machine learning OCR.

The largest existing challenge is improving information recall. While context generation generally performs well, question and answering has mixed performance based on the type of question asked. Nuclear specific questions often lead to poor answers as the models are unable to understand nuclear-specific vocabulary. Two potential improvements are hypothesized. First, base question and answering model (i.e., SciBERT) could be fine-tuned on a nuclear specific text corpus. A similar approach has already been demonstrated by Burke [43]. Secondly, the down-stream task (i.e., question and answering) could be trained on a modified SquadV2 dataset using the salt and pepper technique described by Wilson [44]. Both approaches would improve the model's ability to understand nuclear concepts.

There is work remaining on the robotic side as well. The in house custom datasets need to be expanded, as they are currently only on the order of hundreds of images. Training needs to be expanded from drums only to drums and seals, and we need to integrate optical character recognition for seal identification. The team is also working on a seal pose estimation and grasp planning pipeline, so that Spot can autonomously test the physical integrity of seals. The interface between the on-device app and the robotic platform needs to be fully fleshed out and defined. Finally, the inspection behavior trees need to stitch these capabilities together to enable initial demos of flexible autonomous inspection and inspector teaming.

5. Conclusions

The current machine learning capabilities of the on-device application and robotic companion of the INSPECTA platform are presented. Several key capabilities have been integrated to support the initial seal examination use case. A key focus of this project is the integration of extisting, off-the-shelf, pretrained models. Satisfactory performance is largely met for tasks that are not specific to nuclear safeguards such as the OCR and speech recognition capabilities. However, performance for nuclear specific tasks, such as information recall, needs to be improved through fine tuning on domain specific datasets. Existing models for the robotics platform, such as object detection, have already been adapted for the nuclear specific task of canister and seal recognition. Ongoing efforts are being conducted to improve both platforms with a goal for an integrated demonstration in 2024.

6. Acknowledgements

The authors would like to acknowledge and thank the U.S. National Nuclear Security Administration (NNSA) Office of Defense Nuclear Nonproliferation R&D Safeguards portfolio for funding and supporting this research.

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (DOE/NNSA) under contract DE-NA0003525. This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive,

paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan.

References

[1] International Atomic Energy Agency, IAEA Safeguards Glossary, 2022 Edition.

[2] Xamarin | Open-source mobile app platform for .NET.
URL https://dotnet.microsoft.com/en-us/apps/xamarin

[3] O. developers, Onnx, https://onnx.ai/, version: 1.41.1 (2023).

[4] O. R. developers, Onnx runtime, https://onnxruntime.ai/, version: 1.15.1 (2023).

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, Advances in neural information processing systems 32 (2019).

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[7] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., Transformers: State-of-the-art natural language processing, in: Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations, 2020, pp. 38–45.

[8] B. Dynamics, Spot - the agile mobile robot (2023).
URL https://www.bostondynamics.com/products/spot

[9] D. Huamanchahua, D. Yalli-Villa, A. Bello-Merlo, J. Macuri-Vasquez, Ground robots for inspection and monitoring: A state-of-the-art review, in: 2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), 2021, pp. 0768–0774. doi:10.1109/UEMCON53757.2021.9666648.

[10] W. Echelmeyer, A. Kirchheim, E. Wellbrock, Robotics-logistics: Challenges for automation of logistic processes, in: 2008 IEEE International Conference on Automation and Logistics, 2008, pp. 2099–2103. doi:10.1109/ICAL.2008.4636510.

[11] F. Niroui, K. Zhang, Z. Kashino, G. Nejat, Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments, IEEE Robotics and Automation Letters 4 (2) (2019) 610–617. doi:10.1109/LRA.2019.2891991.

[12] W. Yan, Y. Pan, J. Che, J. Yu, Z. Han, Whole-body kinematic and dynamic modeling for quadruped robot under different gaits and mechanism topologies, PeerJ Computer Science 7 (2021) e821.

[13] B. Dynamics, Spot SDK - spot documentation (2023).
URL https://dev.bostondynamics.com/

[14] Nvidia, Deploy ai-powered autonomous machines at scale (2023).
URL https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/

[15] S. Team, Silero vad: pre-trained enterprise-grade voice activity detector (vad), number detector and language classifier, https://github.com/snakers4/silero-vad (2021).

[16] A. Baevski, Y. Zhou, A. Mohamed, M. Auli, wav2vec 2.0: A framework for self-supervised learning of speech representations, Advances in neural information processing systems 33 (2020) 12449–12460.

[17] R. Prenger, R. Valle, B. Catanzaro, Waveglow: A flow-based generative network for speech synthesis, in: ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 3617–3621.

[18] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. Oord, S. Dieleman, K. Kavukcuoglu, Efficient neural audio synthesis, in: International Conference on Machine Learning, PMLR, 2018, pp. 2410–2419.

[19] T. Shibuya, Y. Takida, Y. Mitsufuji, Bigvsan: Enhancing gan-based neural vocoders with slicing adversarial network, arXiv preprint arXiv:2309.02836 (2023).

[20] V. I. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions and Reversals, Soviet Physics Doklady 10 (1966) 707.

[21] J. Ellis, GitHub - jeremy-ellis-tech/Xamarin.Android.OpenCV: C# bindings for the OpenCV Android SDK (2017).
URL https://github.com/jeremy-ellis-tech/Xamarin.Android.OpenCV

[22] Y. Baek, B. Lee, D. Han, S. Yun, H. Lee, Character region awareness for text detection, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 9365–9374.

[23] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556 (2014).

[24] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, Springer, 2015, pp. 234–241.

[25] EasyOCR, original-date: 2020-03-14T11:46:39Z (Oct. 2023).
URL https://github.com/JaidedAI/EasyOCR

[26] B. Shi, X. Bai, C. Yao, An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition, IEEE transactions on pattern analysis and machine intelligence 39 (11) (2016) 2298–2304.

[27] S. Das, K. Ma, Z. Shu, D. Samaras, R. Shilkrot, Dewarpnet: Single-image document unwarping with stacked 3d and 2d regression networks, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 131–140.

[28] G.-W. Xie, F. Yin, X.-Y. Zhang, C.-L. Liu, Dewarping document image by displacement flow estimation with fully convolutional network, in: Document Analysis Systems: 14th IAPR International Workshop, DAS 2020, Wuhan, China, July 26–29, 2020, Proceedings 14, Springer, 2020, pp. 131–144.

[29] M. Zucker, Page dewarping (Aug. 2016).
URL https://mzucker.github.io/2016/08/15/page-dewarping.html

[30] M. Zucker, GitHub - mzucker/page_dewarp: Text page dewarping using a "cubic sheet" model.
URL https://github.com/mzucker/page_dewarp

[31] C. Zhu, R. H. Byrd, P. Lu, J. Nocedal, Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization, ACM Transactions on mathematical software (TOMS) 23 (4) (1997) 550–560.

[32] C. Souza, A. Kirillov, M. D. Catalano, A. contributors, The accord.net framework (2014). doi:10.5281/zenodo.1029480.
URL http://accord-framework.net

[33] M. Li, T. Lv, J. Chen, L. Cui, Y. Lu, D. Florencio, C. Zhang, Z. Li, F. Wei, Trocr: Transformer-based optical character recognition with pre-trained models, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 37, 2023, pp. 13094–13102.

[34] T. Kudo, J. Richardson, Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, arXiv preprint arXiv:1808.06226 (2018).

[35] The Project Gutenberg eBook of Romeo and Juliet, by William Shakespeare.
URL https://www.gutenberg.org/files/1513/1513-h/

1513-h.htm
[36] Elasticsearch, original-date: 2010-02-08T13:20:56Z (Oct. 2023). URL https://github.com/elastic/elasticsearch
[37] J. Johnson, M. Douze, H. Jégou, Billion-scale similarity search with GPUs, IEEE Transactions on Big Data 7 (3) (2019) 535–547.
[38] V. Sanh, L. Debut, J. Chaumond, T. Wolf, Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, arXiv preprint arXiv:1910.01108 (2019).
[39] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, et al., Ms marco: A human generated machine reading comprehension dataset, arXiv preprint arXiv:1611.09268 (2016).
[40] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, arXiv preprint arXiv:1609.08144 (2016).
[41] I. Beltagy, K. Lo, A. Cohan, Scibert: A pretrained language model for scientific text, arXiv preprint arXiv:1903.10676 (2019).
[42] P. Rajpurkar, R. Jia, P. Liang, Know what you don't know: Unanswerable questions for squad, arXiv preprint arXiv:1806.03822 (2018).
[43] L. Burke, K. Pazdernik, D. Fortin, B. Wilson, R. Goychayev, J. Mattingly, Nukelm: Pre-trained and fine-tuned language models for the nuclear and energy domains, arXiv preprint arXiv:2105.12192 (2021).
[44] B. Wilson, K. Duskin, M. Subramanian, R. Goychayev, A. M. Zuniga, Artificial judgement assistance from text (ajax): Applying open domain question answering to nuclear non-proliferation analysis, ESARDA Bulletin (2021).
[45] J. Redmon, S. K. Divvala, R. B. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, CoRR abs/1506.02640 (2015). arXiv:1506.02640.
URL http://arxiv.org/abs/1506.02640
[46] Ultralytics, YOLOv5: A state-of-the-art real-time object detection system, https://docs.ultralytics.com (2021).
[47] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft COCO: common objects in context, CoRR abs/1405.0312 (2014). arXiv:1405.0312.
URL http://arxiv.org/abs/1405.0312
[48] L. N. National Technology & Engineering Solutions of Sandia, limbo, https://limbo-ml.readthedocs.io/, [Accessed 02-12-2023] (2021).
[49] K. Wada, labelme: Image polygonal annotation with python, https://github.com/wkentaro/labelme (2018).
[50] B. Dwyer, J. Nelson, J. Solawetz, Roboflow (version 1.0), https://roboflow.com (2021).

Appendix

Parameters used for the individual capabilities are described in the following sections.

6.1. Voice activity detection

- **Model:** Silero VAD v4 (https://github.com/snakers4/silero-vad)
- **Activation threshold:** Score ≥ 0.25 (score $\in [0, 1]$)
- **Deactivation threshold:** Two sequential scores ≤ 0.25
- **Buffer size:** 32000
  - Type: float (4 byte)
- **Sample frequency:** 16kHz

6.2. Speech recognition

- **Model:** Wave2Vec2.0 (https://huggingface.co/facebook/wav2vec2-base-960h)
- **Buffer size:** 12000
  - Type: float (4 byte)
- **Sample frequency:** 16 kHz

6.3. Real-time OCR

Real-time OCR is used for the seal examination task.

- **Raw image size:** 720x480 (width, height)
- **Detection model:** CRAFT (https://github.com/clovaai/CRAFT-PyTorch)
- **Detection image size:** 160x120 (width, height)
- **Recognition model:** CRNN (https://www.jaided.ai/easyocr/modelhub/)
- **Recognition image size:** Varies depending on text
  - Max size: Nx64 (width, height)
  - Min size: Nx32 (width, height)
    - There is a minimum height requirement based on the convolution window sizes. Segments smaller than this are resized to a height of 32 while maintaining the original aspect ratio.
    - Resizing attempts to hold the same aspect ratio while scaling height to 32 if smaller than 32 and 64 is larger than 64.
- **Image contrast adjustment:** 2.5
  - Applied to both models
- **Acceptance threshold:** 0.85 (value $\in [0, 1]$)

6.4. Offline OCR

Offline OCR is used for the document scanning task. Parameters are the same as the Real-time OCR unless otherwise noted.

- **Raw image size:** 4080x3072 (width, height)
- **Detection image size:** 960x722 (width, height)
- **Image contrast adjustment:** Varies
  - **Detection:** 2.5
  - **Recognition:**
    - With dewarping algorithm: 0, pipeline produces a binary image so no adjustment is performed
    - Without dewarping algorithm: 2.5

6.5. Document dewarping

There are many parameters involved in the document dewarping algorithm. Rather than listing them here, the reader is referred to the code provided here `https://github.com/nshoman/page_dewarp`.

6.6. Information recall

- **Context model:** DistilBERT TAS-B (`https://huggingface.co/sentence-transformers/msmarco-DistilBERT-base-tas-b`)
  - **Tokenizer:** WordPiece
- **Question and answering model:** SciBERT fine-tuned on SquadV2 (`https://huggingface.co/jbrat/SciBERT-squadv2`)
  - **Tokenizer:** WordPiece