Project Report
LSP-382

An Efficient Methodology for the Development and Evaluation of Advanced Embedded AI Technologies: FY23 Information, Computation, and Exploitation Line Supported Program

V. Gleyzer

24 January 2024

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001.

This report is the result of studies performed at Lincoln Laboratory, a federally funded research and development center operated by Massachusetts Institute of Technology.

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

Massachusetts Institute of Technology
Lincoln Laboratory

An Efficient Methodology for the Development and Evaluation of Advanced Embedded AI Technologies: FY23 Information, Computation, and Exploitation Line Supported Program

V. Gleyzer
Group 102

Project Report LSP-382

24 January 2024

This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001.

Lexington **Massachusetts**

This page intentionally left blank.

TABLE OF CONTENTS

This page intentionally left blank.

LIST OF ILLUSTRATIONS

This page intentionally left blank.

LIST OF TABLES

This page intentionally left blank.

1. INTRODUCTION

Many application domains can leverage advanced artificial intelligence (AI) capabilities in challenging size, weight, and power (SWaP) constrained environments, requiring high-performance embedded solutions. Commercial and academic progress in the embedded AI field is moving rapidly with the development of new specialized processors and AI accelerators, as well as device-aware training and compute-optimized model compressions techniques. However, for large entities such as the Department of Defense (DoD), understanding the ultimate performance gain from the integration of these new technologies to business-critical or mission-critical applications is challenging. The algorithmic performance is typically highly dependent on the exact hardware platform constraints, specific application requirements, and access to representative training datasets. Without this context, published results from general community benchmarks may provide a great starting point to identify potential solutions; however, a more targeted approach is necessary to elucidate their impact on real-world applications.

This document provides a high-level overview of a research and development methodology which leverages a comprehensive framework to build and support an environment for efficient evaluation and design of AI technologies targeting embedded applications. More specifically, by utilizing containers, standard instrumentation utilities, and interface abstractions, it allows users to quickly develop and integrate targeted application test suites in order to jointly evaluate both embedded hardware processors and AI algorithms in the context of any application-of-interest. Because of the modular nature of the architecture, as new models or hardware accelerators become available, they can easily be integrated and evaluated. Since this evaluation is performed on the hardware-of-interest (e.g., machine learning accelerators, central processing units [CPUs], graphic processing units [GPUs], etc.) and uses relevant datasets and high-level application metrics, the results provide meaningful insight on the overall impact of these technologies.

This page intentionally left blank.

2. EMBAI FRAMEWORK OVERVIEW

This section describes several overarching goals for the framework and identifies potential users who can leverage and benefit from the proposed methodology. Next, it introduces the architecture and offers some high-level discussion on how different architecture components enable and relate to different aspects of the proposed objectives.

2.1 GOALS

The EmbAI framework enables development, execution, deployment, coordination, and monitoring of test suites across a myriad of hardware platforms and machine learning software.

The framework provides:

- rapid evaluation and integration of latest hardware accelerators and processors, machine, learning frameworks, algorithms, and models for a key set of representative application;
- simple access to the environment for users of varying levels of expertise;
- experiment reproducibility, reusability, and portability;
- result storage, visualization, and exploration tools; and
- common reusable infrastructure and tools across hardware platform.

2.2 USER CONSIDERATIONS

One of the EmbAI key goals is to help users with varying levels of experience to quickly evaluate and understand the trade-off space of algorithm and hardware selection for their specific application of interest. Based on the experience level and evaluation criteria, there are multiple use-cases of how users can leverage the EmbAI environment. The core philosophy is that advanced users are able to contribute low-level capabilities and through abstraction and well-defined interfaces, these capabilities can be made available to all of the users of the environment. The use-cases described below are listed in the order of complexity, where the first allows evaluation and development with little to no experience of embedded development, while the last allows advanced users to quickly integrate new hardware, algorithms, and applications.

2.2.1 Application-Specific Technology Selection

The most important insight for doing technology selection for a new project is understanding the performance for all algorithm and hardware combinations. This knowledge allows the user to choose the optimal set that satisfies all application constraints. These constraints can come in different forms. For

example, the application may be required to complete its critical processing loop in several milliseconds to keep up with real time requirements, or be able to run on a processor that limits its power consumption to maintain battery endurance. The key for this technology selection process is the ability to run and evaluate all of the combinations of application parameters, representative hardware, models, and datasets.

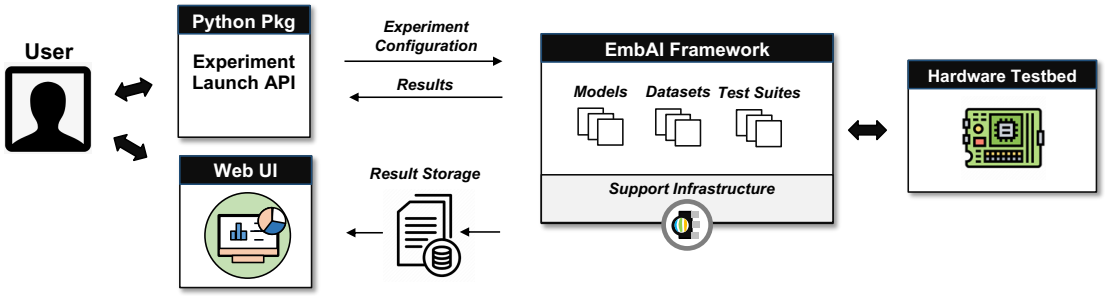

Figure 1. Technology selection use-case.

As shown in Figure 1, the EmbAI environment provides this capability through a simple Python-based experiment launch interface. The interface defines all relevant high-level applications options and allows the user: 1) to define a range of ***relevant*** parameters for a test experiment, 2) launch the experiments on the hardware of interest, and 3) analyze or visualize the performance results.

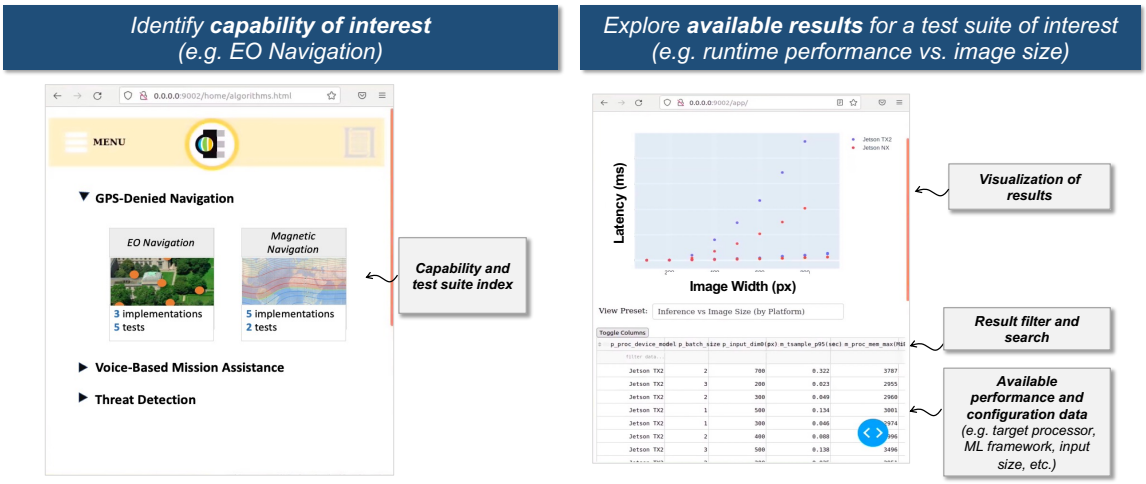

Figure 2. Result exploration User Interface concept.

Alternatively, using the same Python interface, the EmbAI infrastructure can be setup to automatically sweep ***all*** available combination of application parameters a priori. This information is stored in a persistent database. For users who are only interested in understanding general performance trends and trade-offs across different technology combinations and may not have any programming experience, the data is made available through a prototype web-based interface that can enable simple exploration of the results.

In summary, this mode of interaction only leverages existing EmbAI resources, such internally supported model implementations, integrated hardware processing boards, and ML frameworks. It doesn't require any specialized knowledge of the underlying infrastructure and provides simple intuitive tools to visualize and synthesize trends, run application parameter sweeps, and perform general technology trade-off space analysis.

2.2.2 New Model Research and Development

As illustrated in Figure 3, since EmbAI allows the development of a comprehensive suite of relevant evaluation pipelines, adding a new model only requires that it is defined in one of the ML learning frameworks supported by the target evaluation pipeline. These models can then be loaded and deployed by the EmbAI framework where they can be evaluated in a standard way. The results of the experiments can be imported and easily compared to any other potential alternatives.

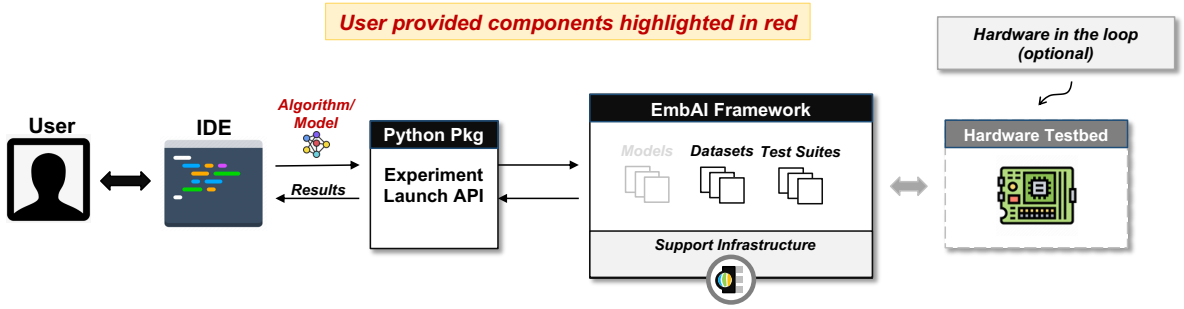

Figure 3. New model development use-case.

2.2.3 New Test Suites and Applications

All existing test suites provide artifacts necessary for enabling a comprehensive test environment, including: high-level application parameters, evaluation utilities, datasets, baseline model implementations and deployable test harnesses. They leverage and build upon existing EmbAI infrastructure, which provides common capabilities such as test integration scaffolding, platform instrumentation, and deployment tools.

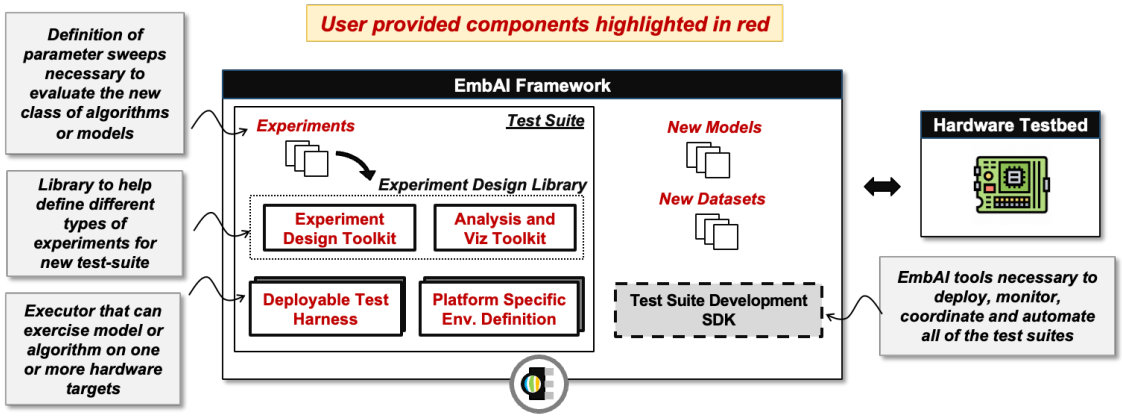

Figure 4. Components necessary for a new test suite, applications, or ML framework.

As shown in Figure 4, developing new test-suites requires the development of these components.Once integrated, however, the new test-suite can seamlessly be launched, used and extended by all users of the EmbAI environment.

2.2.4 New Hardware Platforms and Machine Learning Frameworks

To allow for broad support, the EmbAI environment provides an abstraction-layer and common interfaces which allow the experiments and application-level parameters to be agnostic to the underlying implementation, including the hardware or the ML framework. Adding support for new platforms and frameworks can come in different forms. If the hardware supports a standard operating system and containerization tools, then simple addition of the hardware to the EmbAI test network is enough to run all existing EmbAI experiments and instrumentation utilities in order to collect new performance results across any existing applications with little to no changes. This approach works well for general purpose hardware.

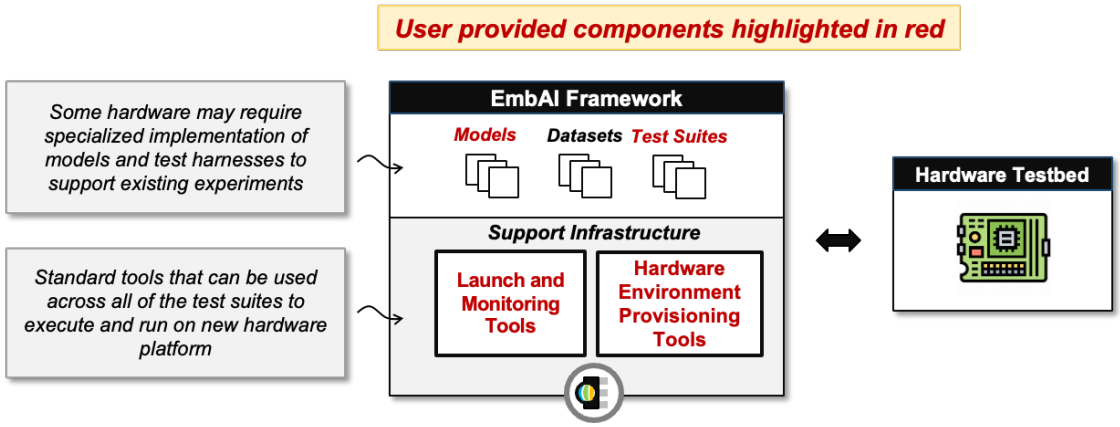

Figure 5. Adding new hardware use case.

Typically, most hardware vendors targeting AI application support standard model definition formats and ML frameworks. If these frameworks are already integrated with EmbAI, the test suite implementations can be reused for evaluation of the new hardware. However, some exotic accelerators require custom machine learning frameworks or drivers. As long as the user provides basic launch and instrumentation utilities and test artifacts which conform to the common EmbAI configuration and results interfaces, these platforms can be easily be integrated with the rest of the environment as well. The new components that need to be provided by the user are illustrated in Figure 5.

2.3 HIGH-LEVEL ARCHITECTURE OVERVIEW

To accelerate development and integration, each supported application test-suite needs to provide a set of standard development packages that define the experiment parameters space and tools necessary to run, execute and evaluate new experiments. The actual runtime setup and configuration of the underlying hardware is completed by a set of common helper components of the EmbAI infrastructure. These general tools take care of all provisioning, configuration, and monitoring of the actual experiment. Once results are available, they are parsed into a data model representation which is common across all of the experiments of the same application type. This shared representation allows for a broad analysis of trends and simple visualization and plotting of results across different hardware, machine learning frameworks, and models.

Each test run is defined by a configuration which captures all of the relevant application-level experiment parameters necessary to configure and evaluate the full system. Certain parameters are application specific, while others, such as model type and dataset, are shared across all of the applications. The experiments define parameter sweeps that probe the parameter space and allow either algorithmic or runtime performance evaluation.

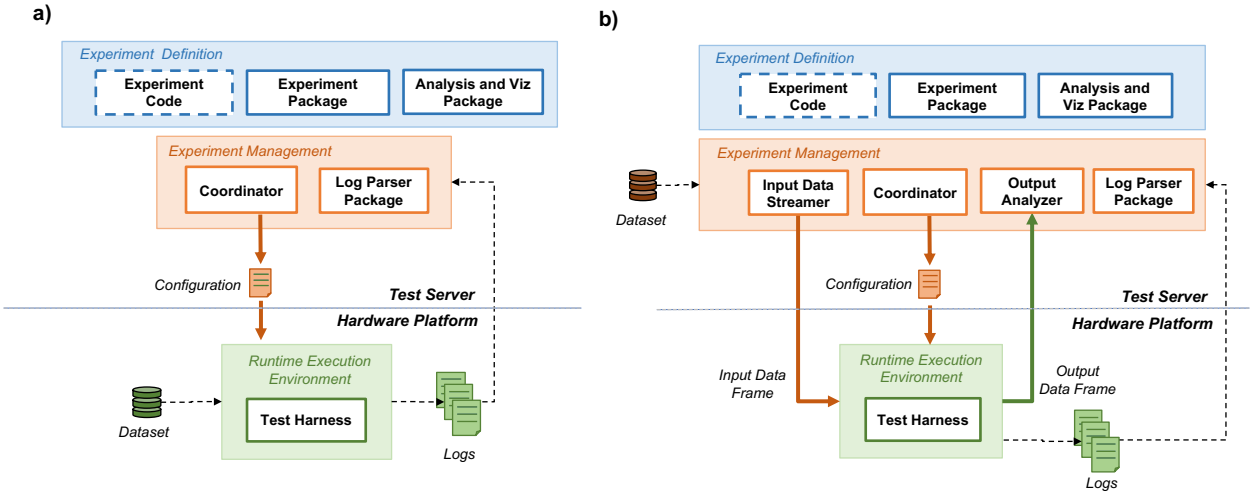

Figure 6. Architecture components that enable deployment and execution of experiments; a) dataset stored locally on the device, b) dataset streamed from the test server.

As illustrated in Figure 6, depending on the device resource availability, the datasets can be read directly from local storage of the device under test, or can be streamed from the test server during the evaluation. The generated logs are sent back to the test server for integration into the final data model of the results. The logs contain resource utilization data, runtime performance, and any other relevant experiment output.

The actual execution is performed on the target device using a test harness. It implements a minimal pipeline necessary to evaluate a ML model leveraging a specific ML framework and supporting a set of hardware platforms. The test harness is deployed in a "Runtime Execution Environment." The environment provides a simple, self-contained deployment unit with all of the instrumentation, drivers, and dependencies necessary to support the execution of the test harness.

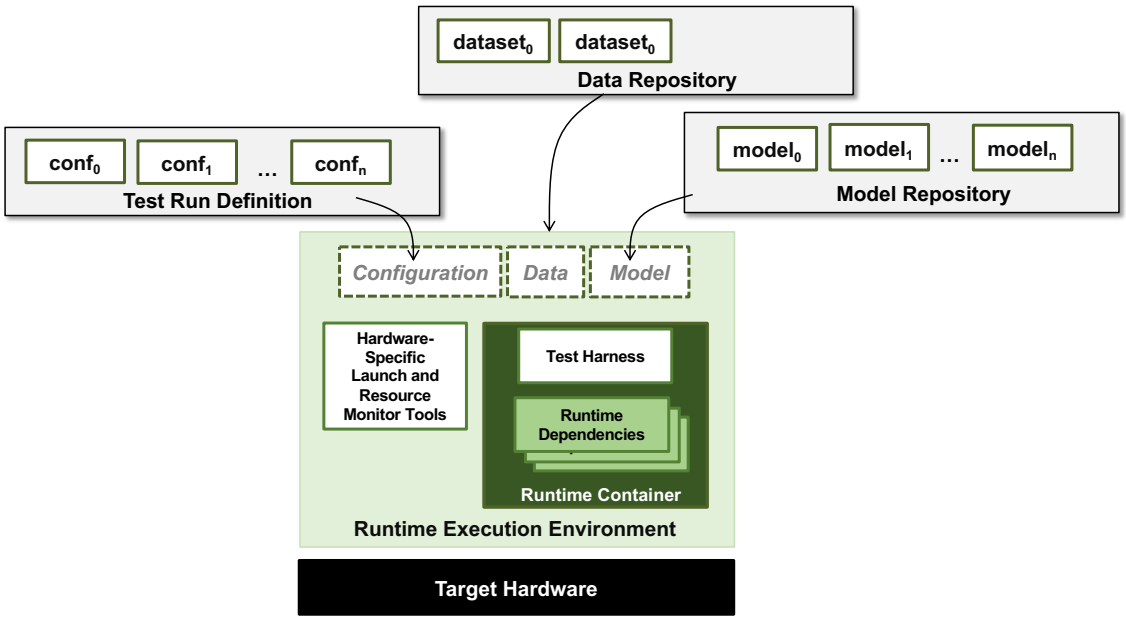

Figure 7. Runtime Execution Environment.

As shown in Figure 7, other artifacts, such as experiment configuration, datasets and model, are populated during experiment setup by the experiment coordinator.

The rest of the section is dedicated to describing key aspects and consideration for different parts of the framework.

2.3.1 Containerization

There are several logistic challenges of developing and running reproducible benchmarks across a "fleet" of embedded hardware platforms from different vendors:

1. The underlying software ecosystem for each platform is constantly changing as patches and versions of different software components and dependencies are released. These updates are usually asynchronous since they are released by different organizations and developers. Many of the changes may not be backwards compatible, which can break build or runtime environments.

2. The development and the runtime environments across embedded hardware systems are typically not compatible and require vendor-specific tools and libraries.

3. Certain hardware configuration, such as clock frequency or maximum power, can have significant impact on the performance benchmark result. Documenting and maintaining the

environment that is used for the evaluation is critical for experiment reproducibility and key understanding of the data.

In order to address these challenges, the EmbAI infrastructure is leveraging containerization technology. The use of containers has become an industry standard for software deployment. They provide a systematic approach for creating reproducible environments that can be created, instantiated and archived. They can capture all of the dependencies, including: system libraries, application packages, configurations and executables.

The EmbAI infrastructure defines two types of containers: development and runtime. The development containers have all of the development and build tools necessary for compilation and packaging to a specific platform, while runtime containers have the minimum runtime dependencies required to support the execution of benchmarks. The two types of containers allow for simple multi-stage build strategy, where the artifacts generated in the development containers can be copied over to containers derived from the runtime containers.

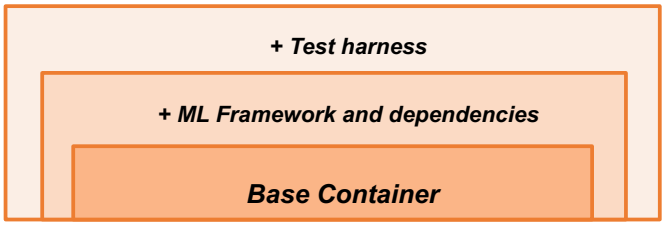

Figure 8. Container hierarchy.

As seen in Figure 8, to help with container reuse and maintenance, the infrastructure provides a simple hierarchy of containers which build upon each other. At the core, the set of base containers are designed to hold common packages and configurations that can be reused across all other containers. They provide the foundation and capture the minimal requirement and instrumentation utilities necessary for all EmbAI environments. They can be custom built or extend upon existing containers provided by the hardware vendors.

Simple access to fully-configured, proven ML environments that can be launched across different hardware platforms is a key feature of the EmbAI infrastructure. The next level of containers provides this type of baseline capability for individual Machine Learning frameworks supported by each hardware. They can be reused across test suites and are preconfigured with any typical dependencies that are used during model development. The definition files are versioned and are updated to track the latest frameworks and their updates.

The final layer of containers is used for deployment of the test suites. They are specialized to launch the test harnesses and have standard volume mounts required to interact with different datasets and

experiment parameters. Since not all embedded systems have Operating Systems or standard support for containers, the deployment containers can be of two forms.

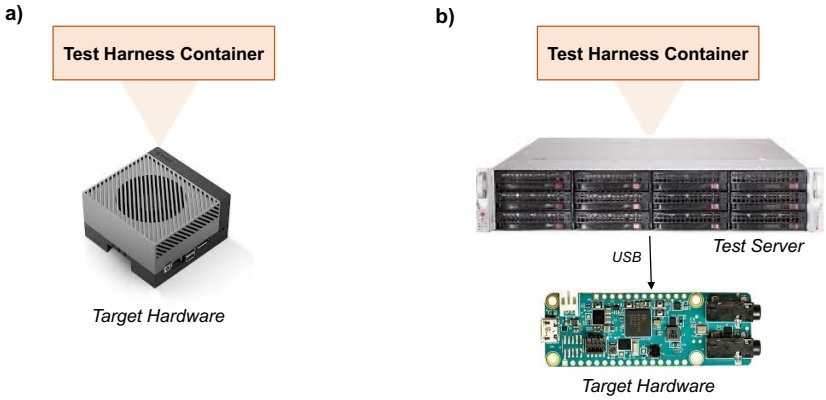

Figure 9. Example of two deployment container types: a) container can execute directly on the target hardware; b) the test container is executed on a test server which then acts as a proxy and interacts with the target hardware to configure and instruments perform any test.

As illustrated in Figure 9a, the first type is a standalone deployment container which can run directly on the target hardware. This form is used to execute test harnesses on all platforms that support container-based execution. The second type, shown in Figure 9b, runs on a test server and acts as a surrogate for the actual target hardware. It encompasses tools and utilities necessary to program, instrument and interact with the target boards. It's important to note that from the perspective of the EmbAI launch infrastructure, both of these container types run and generate standard data products, and therefore, the rest of the system is agnostic to the actual container implementation.

2.3.2 Models

Depending on the ML framework, the model definition can take many forms. Some framework can import standard formats, such ONNX; however, others may require framework-specific syntax, programming language, or model storage. Furthermore, the specific features and capabilities vary greatly across the individual frameworks and depend on the target ML community and applications.

In order to allows support for a broad range of boards, frameworks, and vendors, the EmbAI framework explicitly doesn't prescribe any specific format. It defines a standard way to version, package, release, and deploy their implementation. However, models implemented in standard or framework-agnostic ways are preferred since these implementations can be deployed and used across multiple different frameworks and test harnesses.

All source code and environment necessary to generate a model itself is managed via a Git-based version control system along with a standard test, release, and versioning procedure. This approach has several benefits:

- Because all models and source code are versioned within the EmbAI environment, the implementation can be unequivocally identified and regenerated as necessary. This capability allows for evaluation and verification of results retroactively, which can be tremendously useful when trying to explain unexpected results or verifying implementation correctness. Furthermore, some model deployment formats, such as TensorRT engines, are not meant to be human readable, and therefore, after they have been generated are hard to examine or verify for correctness. Ability to reach back to the code that created the model can be critical for analysis.

- Every framework has many different generation flags and parameters that can have significant effect on the final output model. Versioning the build environment along with the model captures these parameters for completeness. For example, for ONNX models that were defined and exported from PyTorch, the repository would include a conda based environment file, original PyTorch model, export parameters and any other tools and utilities that are required to generate the ONNX model.

- As versions of frameworks are updated, the build environment can be updated as well to support them. The test targets can then be leveraged to perform regression testing in order to verify model correctness before final release.

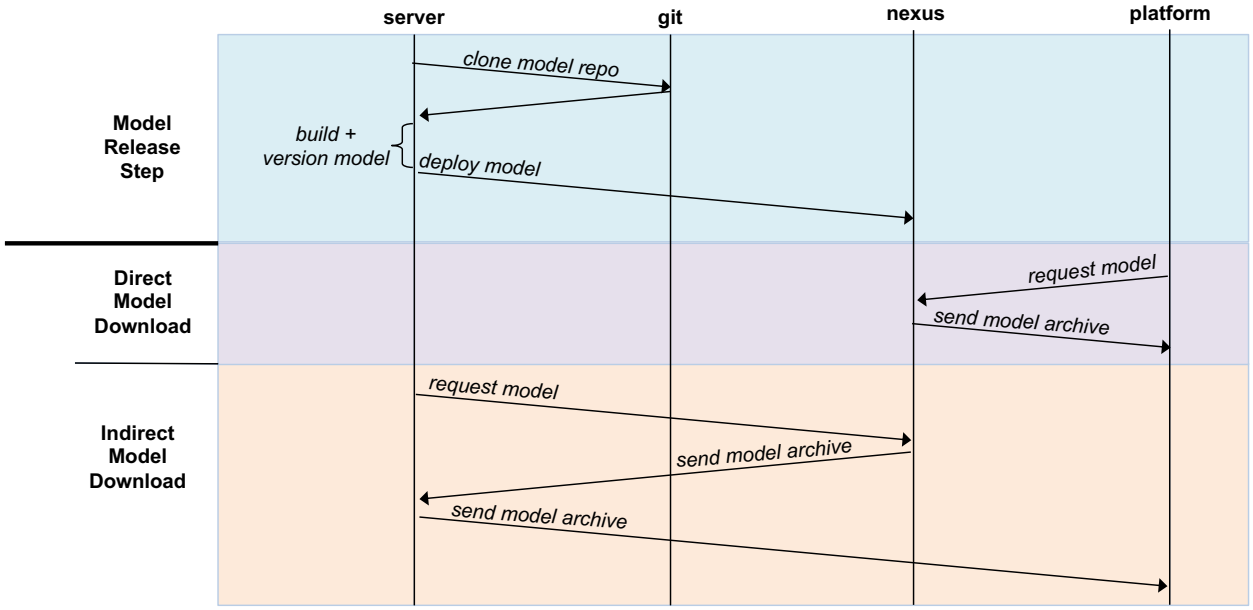

Figure 10. Model deployment options.

The model artifacts are stored in a central binary artifact repository. The EmbAI infrastructure leverages Sonatype Nexus, but, theoretically, any binary management system can be supported. As demonstrated in Figure 10, after the models are generated, they are packaged and deployed to the central artifact repository. They can then be accessed directly through the network from the different hardware platforms. In order to support platforms that do not have network connectivity or are severely resource limited, the model can be downloaded indirectly. The model is first downloaded to the test server, which can then transfer the model to the test hardware using protocols and interfaces that are supported by the target.

2.3.3 Datasets

Access to application-specific datasets is an important part of the infrastructure. Even though there are many different datasets across different application and benchmarks, there is very little consensus regarding storage formats. To help minimize the need of writing dataset specific parsers, the data can be pre-processed to match organization or access patterns that are common across multiple datasets which are used to evaluate the same class of algorithms or models. All source code for the pre-processing, versioning and release of the datasets is captured in a Git repository. This approach provides a clear log of the actual data that is used in all of the experiment and allows for data to be recovered for post-experiment verification purposes. Similar to the models, the release artifacts are pushed to a central binary artifact repository, where they can be accessed during experiment setup and configuration phases.

2.3.4 Test Harnesses

A test harness is a program that implements the minimal pipeline required to evaluate inference performance for a model or an algorithm. This pipeline is deployed to the target hardware and is executed as part of the experiment. The typical capabilities of a test harness include ability to ingest application specific datasets, apply simple transforms, load inference model, perform inference, and provide instrumentation.

An ideal runtime harness implementation would execute on multiple hardware platforms and support many machine learning frameworks. This approach would maximize code reuse while minimizing maintenance cost. However, there are many practical reasons that may limit the usability of a specific test harness implementation to a constrained set of supported platforms, for example:

- requirement for a specific programming language (e.g., Python, C++, C, Julia),
- platform-specific accelerators interfaces (e.g., CUDA, OneAPI, etc.), and
- conflicting software dependencies.

Since it is not feasible to have a single test harness that can provide such broad cross-platform support, EmbAI infrastructure allows for a family of test harnesses that can be re-targeted to a broad set of hardware. Hardware vendors tend to optimize support for one or two popular ML frameworks to help existing ML communities leverage their processors. Thus, test harnesses centered around these popular ML frameworks would ensure broad coverage across many vendors. For example, a TensorRT-based implementation can be deployed on all platforms supporting the TensorRT ecosystem, while PyTorch-based test harnesses can be deployed to any device with PyTorch support.

The test harnesses for a specific class of models are interchangeable and must support a standard set of experiment configuration, input, and output interfaces. Every implementation is not required to implement the full space of all possible configuration parameters. Even the same test harness deployed on different systems may limit the set of supported parameters. As an example, a harness may only be able perform inference for input images smaller than 300x300 pixels on one platform, but easily support 600x600 pixels images on a system with larger memory capacity.

Even though the harness itself can provide feedback that it does not support a specific configuration, the current EmbAI infrastructure does not have a standard methodology for encoding specific configuration constraints for a test-harness running on specific platforms. This functionality was out of scope of this effort but would be a good feature addition in the future. It would allow the system to filter invalid configurations without attempting to instantiate the full test harness on the target platform and generating an unsupported configuration error.

2.3.5 On-Device Caching

In the embedded context, another important consideration is the memory capacity and communication bandwidth of the devices that are being evaluated. Since datasets, models and test harnesses can potentially consume a substantial amount of memory, moving and storing these different artifacts directly on the devices as part of provisioning for a particular experiment configuration can be challenging. For the execution flow illustrated in Figure 6a, the EmbAI environment leverages on-device caching and configuration reordering to maximize reuse and minimize the time it takes to download and run the experiments. The strategy allows a practical tradeoff during a parameter sweep between

- constantly downloading all necessary objects with every configuration, and
- analyzing all of the configuration and grouping them in an order that allows maximum reuse of the on-device artifacts.

As an example, if the full parameter sweep for a vision test suite requires all combinations of two large datasets and five different image sizes. The experiment coordinator would order the execution such that each dataset is downloaded to the device only once. All of the configurations requiring that dataset would then be prioritized for execution before the second dataset is considered. The data would be cached locally, which would minimize the number of datasets that are on the device at one time, and ensure that it is available for all of the required configurations.

2.3.6 Experiment Logs and Platform Instrumentation

Each test harness collects critical runtime statistics. They are captured and stored in a standard format across all of the test harnesses of the same type. The format is versioned, which allows it to be upgradable while providing a simple way to recreate result analysis from older collection campaigns without re-generating log artifacts using the latest test apparatus. The log contains the configuration as well as the performance summary for the test run. All statistics across a single dataset are summarized using distribution statistics, such as median, maximum, minimum, and etc. They can be used to understand the spread and help inform worst-case and best-case analysis.

Alongside the test harness, the system is instrumented to collect vital resource statistics such as processor utilization and memory consumption. The resources utilization collection strategy is unique for the hardware platform. Once the instrumentation utilities are implemented, they can be shared across all of the test suites. The results are stored in a common format, maximizing re-use, and simplifying the process of deploying new applications.

2.3.7 Result Data Models and Analysis

In order to enable broad trend analysis across a heterogenous set of hardware, each application type defines an EmbAI result data model which unifies all critical information from the experiment. The logs

collected are parsed, and the results are used to populate the data model fields. Since all experiments are mapped to the same result model, they can be analyzed together to provide global insight.

The information can be stored in a document database, such as ElasticSearch, or returned directly to the user after the experiment is complete. Since the data model is also versioned, different helper and analysis utilities can be developed in tandem to assist aggregation, visualization and analysis. These tools can be upgraded as the data model evolves in order to support different type of hardware and new data fields.

2.4 SOURCE CODE ORGANIZATION

To minimize dependencies and allow for independent development of different components, the EmbAI infrastructure is organized in a suite of loosely coupled repositories.

2.4.1 Repository Naming Convention

To help navigation, repositories are named using a simple hierarchical naming convention. The format follows a hierarchical dot notation, where the first component indicates the type of repository. Table 1 describes five different types of repositories.

Table 1

Repository Types

Name	Description	Examples
sdk	Software Development Kit for development of new test suites and experiments	*sdk.python*
testsuite	Source code for developing experiments for a specific test suite	*testsuite.feature-detection*
models	Group of related models for a particular framework	*models.pytorch* *models.opencv*
framework	Provisioning and instrumentation utilities targeting a specific hardware family	*framework.x86_64* *framework.jetson*
examples	Example code demonstrating basic use of an ML framework or any other relevant technology. These repositories can be used for reference or as a test driver to verifying basic functionality for runtime or development environments.	*examples.tensorflow* *examples.docker*
data	Dataset pre-processing code	*data.images*
external	EmbAI-local fork of an external repository	*external.tflite-micro*

2.4.2 Python Software Development Kit

The core functionality of the EmbAI environment is organized into a Software Development Kit (SDK). It contains a collection of foundational packages that can be leveraged to develop new front-end components such as test suites, evaluation environments and high-level experiments. To match the current industry trends of AI community, the SDK has been developed in Python; however, support for other language can be added in the future.

Every package is individually installable via the standard Python package manager, which captures and tracks any internal and external dependencies. This management approach mimics other large SDKs, such as Azure SDK4. It minimizes the third-party dependency requirements and simplifies deployment of any development environment that solely depends on a small portion of the SDK.

This page intentionally left blank.

3. SUMMARY

This report describes a comprehensive methodology for a research development environment that can provide an efficient way to continuously evaluate new embedded AI technologies, including algorithms and hardware accelerators, as they relate to representative business-critical and mission-critical applications and pipelines. The evaluation capability is not only essential for tracking any existing and up-and-coming technologies from industry and academia, but can be invaluable for identifying technology gaps and ultimately providing feedback for research and development of new solutions. Furthermore, the methodology ensures that any capability that is integrated into the environment can be shared across any number of test suites and user experiments. This flexibility promotes code reuse which ultimately lowers the effort required to support and evaluate new technologies.

This page intentionally left blank.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
24-01-2024	Line Supported Program	

4. TITLE AND SUBTITLE

An Efficient Methodology for the Development and Evaluation of Advanced Embedded AI Technologies

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Vitaliy Gleyzer

5d. PROJECT NUMBER
2236/4301

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02421-6426

8. PERFORMING ORGANIZATION REPORT NUMBER

LSP-382

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Information, Computation, and Exploitation Line at MIT LL

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This document provides a high-level overview of a research and development methodology which leverages a comprehensive framework to build and support an environment for efficient evaluation and design of AI technologies targeting embedded applications. More specifically, by utilizing containers, standard instrumentation utilities, and interface abstractions, it allows users to quickly develop and integrate targeted application test suites in order to jointly evaluate both embedded hardware processors and AI algorithms in the context of any application-of-interest.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	None		