Project Number 101017258

D6.3 Tools for Automated Quality Assurance of EDDI-Supported MRS (Initial Version)

Version 1.0
30 June 2022
Final

Public Distribution

University Of York

Project Partners: **Aero41, ATB, AVL, Bonn-Rhein-Sieg University, Cyprus Civil Defence, Domaine Kox, FORTH, Fraunhofer IESE, KIOS, KUKA Assembly & Test, Locomotec, Luxsense, The Open Group, Technology Transfer Systems, University of Hull, University of Luxembourg, University of York**

Project Partner Contact Information

Aero41 Frédéric Hemmeler Chemin de Mornex 3 1003 Lausanne Switzerland E-mail: frederic.hemmeler@aero41.ch	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany E-mail: scholze@atb-bremen.de
AVL Martin Weinzerl Hans-List-Platz 1 8020 Graz Austria E-mail: martin.weinzerl@avl.com	**Bonn-Rhein-Sieg University** Nico Hochgeschwender Grantham-Allee 20 53757 Sankt Augustin Germany E-mail: nico.hochgeschwender@h-brs.de
Cyprus Civil Defence Eftychia Stokkou Cyprus Ministry of Interior 1453 Lefkosia Cyprus E-mail: estokkou@cd.moi.gov.cy	**Domaine Kox** Corinne Kox 6 Rue des Prés 5561 Remich Luxembourg E-mail: corinne@domainekox.lu
FORTH Sotiris Ioannidis N Plastira Str 100 70013 Heraklion Greece E-mail: sotiris@ics.forth.gr	**Fraunhofer IESE** Daniel Schneider Fraunhofer-Platz 1 67663 Kaiserslautern Germany E-mail: daniel.schneider@iese.fraunhofer.de
KIOS Maria Michael 1 Panepistimiou Avenue 2109 Aglatzia, Nicosia Cyprus E-mail: mmichael@ucy.ac.cy	**KUKA Assembly & Test** Michael Laackmann Uhthoffstrasse 1 28757 Bremen Germany E-mail: michael.laackmann@kuka.com
Locomotec Sebastian Blumenthal Bergiusstrasse 15 86199 Augsburg Germany E-mail: blumenthal@locomotec.com	**Luxsense** Gilles Rock 85-87 Parc d'Activités 8303 Luxembourg Luxembourg E-mail: gilles.rock@luxsense.lu
The Open Group Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium E-mail: s.hansen@opengroup.org	**Technology Transfer Systems** Paolo Pedrazzoli Via Francesco d'Ovidio, 3 20131 Milano Italy E-mail: pedrazzoli@ttsnetwork.com
University of Hull Yiannis Papadopoulos Cottingham Road Hull HU6 7TQ United Kingdom E-mail: y.i.papadopoulos@hull.ac.uk	**University of Luxembourg** Miguel Olivares Mendez 2 Avenue de l'Universite 4365 Esch-sur-Alzette Luxembourg E-mail: miguel.olivaresmendez@uni.lu
University of York Simos Gerasimou & Nicholas Matragkas Deramore Lane York YO10 5GH United Kingdom E-mail: simos.gerasimou@york.ac.uk nicholas.matragkas@york.ac.uk	

Document Control

Version	Status	Date
0.1	Document outline	1 June 2022
0.2	First draft	10 June 2022
0.7	First full draft	17 June 2022
0.8	Further editing draft	21 June 2022
1.0	QA review	30 June 2022

Table of Contents

List of Figures

Executive Summary

This document supports the prototype tools developed in Tasks 6.1 and 6.2 and provides a getting-started guide for those prototypes. The prototype tools implement the requirements, principles, and architecture described in deliverables D6.1 and D6.2.

To this end, D6.3 reports on the required metrics, configurations and tools as well as the design of the related functionalities at subsequent stages of the project.

The document provides information for installing the developed prototype tools and a "how-to" for using these tools, complementing the information provided on the tools Github repositories.

1 INTRODUCTION

This documents provides a summary of the tools developed in work package WP6 during the 18 months of the SESAME project. The purpose of this report is to complement the deliverables D6.1 and D6.2 that describe the theoretical concepts underpinning the techniques developed for assuring learning-enabled components and simulation-based testing, respectively. To this end, the report describes how these tools can be used making explicit reference to their source-code which is available on Github. Accordingly, the user guide for our simulation-based testing tools is provided in Section 2, while the guide for our learning-enabled testing tools is provided in Section 3.

The methodologies and tools described in this report are self-contained. However, as the project progresses the tools will also be extended in order to support the simulation-based testing of EDDIs, as these are further defined during integration work with project partners, especially those working in work packages WP4 and WP5. The simulation-based testing platform that is presented here in Section 2 can be informed by the safety analysis that underlies the construction of the EDDIs (e.g., the construction of fault trees or Bayesian networks) to drive the selection of the safety-critical system components to fuzz. The platform can be used to verify the correct detection and mitigation of failing scenarios by the EDDI components, for example, using its performance metrics to be sure that safe mitigations are engaged. Further, once the EDDI is integrated with the simulation via a compatible interface, communication channels to the EDDI can be tested, since themselves can be subject to fuzzing. This will enable determining how the EDDI responds to these scenarios and whether it can maintain safety and security guarantees.

The DeepKnowledge framework, described further in Section 3 could also be used to evaluate the quality of the EDDIs. Its capabilities could be used in order to assess knowledge generalisation within the EDDIs, assessing the degree to which ML-equipped EDDI components will be able to respond to environmental conditions that are different to the conditions under which they were originally trained.

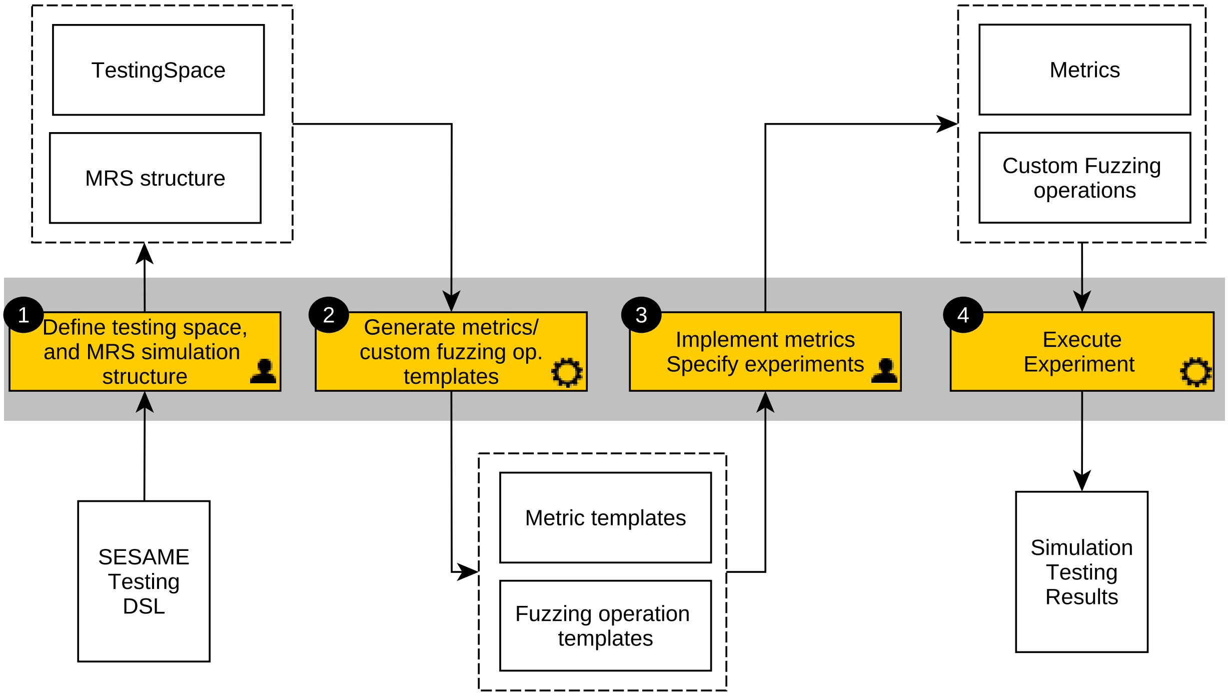


Figure 1: Simulation-based testing methodology

2 Simulation-Based Testing Tools

2.1 Overview

2.1.1 SESAME Simulation-Based Testing

The simulation-based testing platform proposed in the context of WP6 interfaces with the MRS simulation, and according to the currently supplied testing configuration, deliberately alters or manipulates the simulation state in order to assess how the system under test responds to such scenarios. This allows to assess the behaviour of the simulation under such disruptions by means of supplied scenario-specific performance metrics. This report provides an overview of the SESAME simulation-based testing framework, its methodology and architecture, a user guide and example of its application to an MRS case study. The report complements the provided opens-source code and deliverable D6.1.

2.1.2 SESAME Simulation-Based Testing Methodology

SESAME WP6 enables the evolution of effective fuzz testing campaigns that reveal violations of system safety and security requirements. These testing campaigns can lead to failure-inducing scenarios that can be analysed by domain experts and further inform the hardening of the MRS system. Identifying these scenarios using the simulation-based testing framework, entails following the methodology presented in Figure 1. Please note that the four steps of the methodology are labelled according to whether they involve human input or whether they are automated. The steps are summarised below. For further details, please refer to our deliverable D6.2 [6]. An example of its execution is given in Section 2.3:

Step 1: MRS testing engineers instantiate the SESAME testing DSL to produce a model that encodes the fuzzing testing space including the messages of interest, fuzzing operations and timing constraints. The

MRS structure model corresponds to information that are provided by the Executable Scenarios workbench (see deliverable D3.2) and which are retrieved by executing a model-to-model transformation. This MRS model encodes the characteristics of the target mission including the robotic systems, the simulation structure, and the mission requirements. Currently, in the supplied implementation, these two models are combined into a single model resource, and the users must supply this stub scenario information as well. We provide the specification of the testing metamodel DSL in [6].

Step 2: The code generation engine employs the devised models and automatically generates mission requirement metric quantification templates whose instantiation enables to assess whether a mission or safety/security requirements are met, and if not, the extent and impact of the violation. Since the fuzzing operations selected and requirements quantification are mission and system-specific, users are responsible for populating these templates.

Step 3: Users implement the previously defined metric templates, to create the required custom scenario-specific metrics. In particular, they specify via the SESAME DSL particular fuzzing test campaigns, corresponding to the particular experiments which they intend to execute. These constitute a selection of a particular set of fuzzing operations defined in the testing space, and a subset of the defined metrics to use to assess requirement violations. The type of experiments, and any experiment-specific parameters are also specified. For example, with genetic algorithms, experiment-specific parameters such as the number of generations and iterations can be included.

Step 4: The SESAME fuzzing engine consumes the provided model, metrics and selected experiment details and utilises the information from these models in an evolutionary optimisation step, during which it evolves a population of tests. The experiment runner which incorporates the evolutionary algorithm evaluates each test by first dynamically generating the code for a specialised test runner which acts as a middleware, communicating with the low-level simulator via a simulation specific interface, and using any custom supplied metric definitions provided in Step 2 to quantify the impact of the fuzzing test. As a test is run, this information is communicated to the experiment runner and used to guide a multi-objective optimisation process. This process uses genetic operations such as mutation and crossover to create new tests, discarding the worst performing campaigns from the population. This iterative process continues until an experiment-specific termination criterion is satisfied, i.e., either the maximum number of permitted generations is reached, or no improvement occurs over a specified number of evolution rounds.

2.1.3 SESAME Simulation-Based Testing Architecture

The high-level architecture of the simulation-based testing framework is depicted in Figure 2. A primary component of the SESAME simulation-based testing framework is a domain-specific language (DSL), detailed in [6], that supports the specification of the fuzzing space, providing the structure of the available fuzzing operations to be performed.

The DSL is paired with a fuzzing experiment runner, that consumes DSL-compliant models and generates individual test runners. Each test runner is configured to perform a particular Test, with a set of fuzzing operations and activation parameters. The test runner consists of a middleware (instrumented upon Apache Kafka/Flink) and a fuzzing engine that distorts messages according to the test runner configuration. The middleware communicates with the MRS via a platform-specific interface, and operates as a "man-in-the-middle" which routes messages from the MRS via the fuzzing engine. The fuzzing engine will optionally distort, delete or delay the message (according to the test runner configuration) before rerouting it back to the MRS and its intended recipients. The values of custom scenario-specific performance metrics from these tests will be communicated back from the fuzzing experiment runner and stored in the DSL for the given Test.

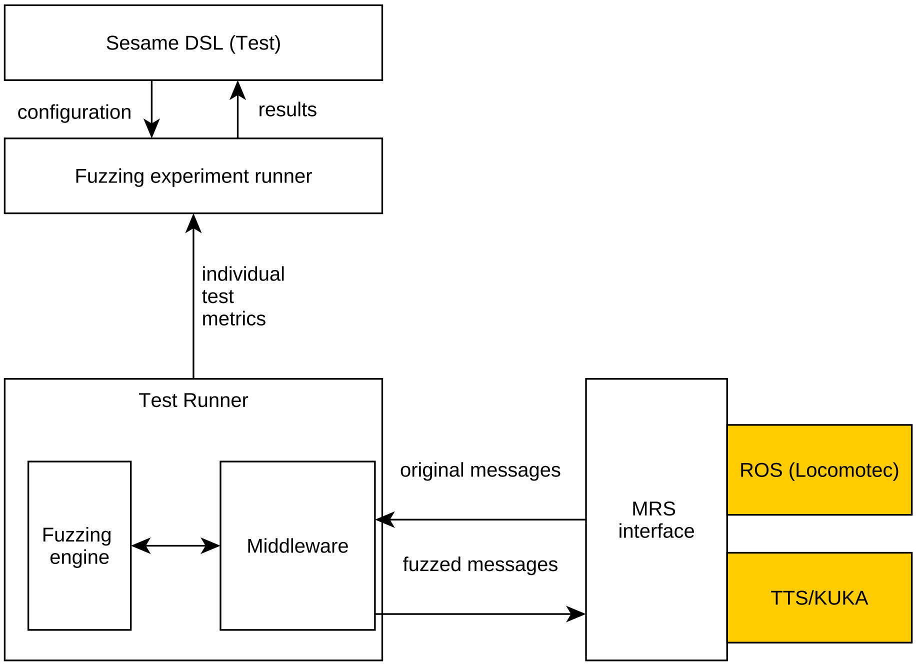


Figure 2: The architecture of the test runner and its runtime connection to the MRS

2.2 User Guide

2.2.1 Introduction

The SESAME simulation-based testing infrastructure is implemented as a set of Java projects and tooling integrated with Eclipse, building upon open-source and widely-used model-driven engineering tools such as the Eclipse Modelling Framework[1], Epsilon[2] and Emfatic[3]. Additional standard Java technologies such as the Maven build tool[4] are used to recompile code components dynamically generated during the execution of the experiments. Apache Kafka[5] and Flink[6] are used for message communication and stream processing, in order to interconnect the MRS simulator, the individual test runners, and the fuzzing engine that manages the experiments.

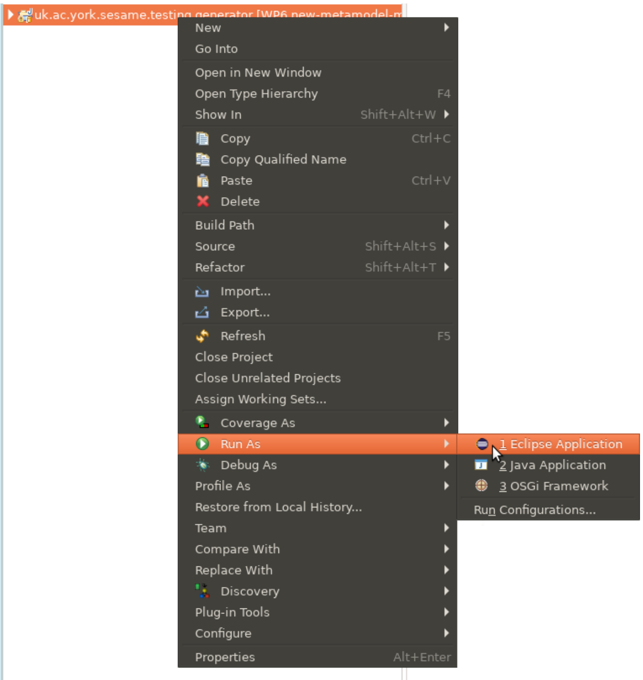
2.2.2 Setting Up Software and Dependencies

The complete installation guide is presented as a README guide within the Github repository at `https://github.com/sesame-project/simulationBasedTesting`. After completing the installation process indicated there, please continue with the user guide below.

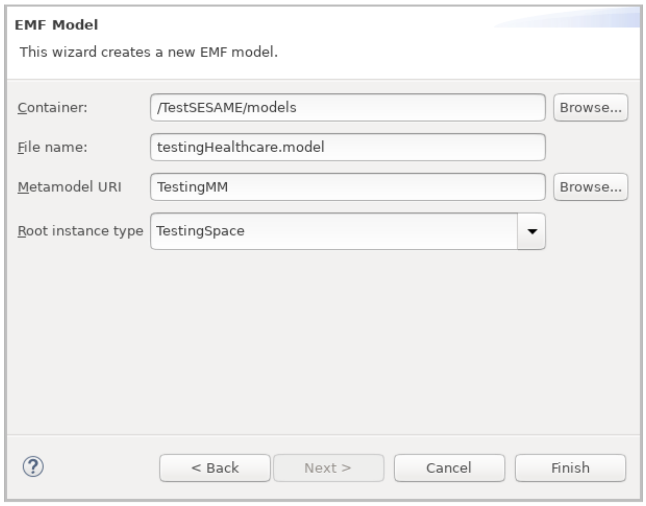
2.2.3 User Guide

In order to use the simulation-based testing platform, the user should invoke a new Eclipse Application, by right-clicking upon the newly imported project **uk.ac.york.sesame.testing.generator** and selecting “Run As”

[1]https://www.eclipse.org/modeling/emf
[2]https://www.eclipse.org/epsilon
[3]https://www.eclipse.org/emfatic
[4]https://maven.apache.org
[5]https://kafka.apache.org
[6]https://flink.apache.org



(a) Run generator project



(b) New model generation

Figure 3: Generation steps for the SESAME simulation-based framework

/ “Eclipse Application”. This will launch a fresh Eclipse instance under which the SESAME automated code generation plugins are available (Figure 3a). (If there is a problem flagged regarding “javax.xml.bind”, ignore it). There may be a delay on the first invocation of this.

Create a new Java project - here, we use **TestSESAME**. Create a folder “models” in it. When creating the project, turn off “Create module-info.java” at the bottom of the project dialog.

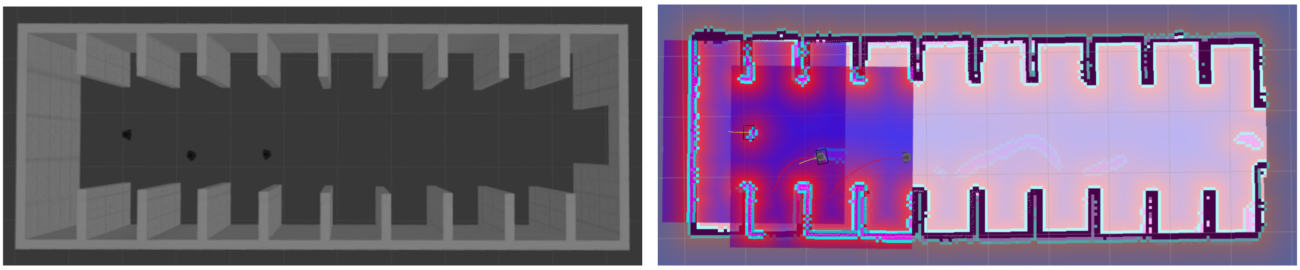
In order to generate a model for the first time in a newly created project, it is necessary to register the metamodels. This can be done by activating the early stage of our wizard, by right-clicking on “SESAME” / “Generate SESAME Code”, as shown in Figure 6a. Then, click Cancel on the dialog box that appears.

Next, create an instance of the testing metamodel. To do this, right click on the folder “models” and select “New” / “Other” / “Epsilon” / “EMF Model” and set up the parameters as shown in Figure 3b. Use “Browse” to find “TestingMM” as the metamodel URI. The model filename can be chosen to fit the scenario that the users are setting up.

Now users can complete the steps specified in the SESAME simulation-based testing methodology. When the model is completed, or when there are any changes to the experiments, right click on the project, and select “SESAME” / “Generate SESAME Code” to regenerate code for metric templates and experiment runners based on it (Figure 6).

After this code generation is performed for the first time, the project structure must be changed to a Maven project. Users should right-click on the project, select “Configure” / “Convert to Maven Project”. Then the user should right-click and select “Maven” / “Update Project” to ensure all the dependencies are updated.

An example for the next stage is given in Section 2.3. This provides an example for setting up a model for a healthcare case study.



(a) Gazebo simulation

(b) rviz visualisation

Figure 4: Healthcare case study example showing three robots in operation visiting rooms

Table 1: Healthcare robots mission requirements

ID	Description
R1	The robots must service all their assigned rooms.
R2	Each robot must return to its starting location for recharging before its battery is depleted and become stranded in the hospital environment.
R3	The robots must fulfil requirements R1 and R2 in the shortest possible time and before T_{end} (the maximum mission end time).

2.3 Example

In order to illustrate the simulation-based testing tools, we present an example MRS case study. The case study chosen is a team of multi-purpose mobile healthcare robots that collaborate to service rooms within a hospital facility. We assume that each of the three robots is pre-assigned to visit a simple sequence of occupied rooms, and to complete a task with a variable completion time within each room (e.g., check patient status or take measurements). Figure 4a depicts the simulation environment in Gazebo, whereas Figure 4b shows an rviz visualisation capturing also the localisation area and navigation path per robot (i.e., showing that two robots are moving towards entering the rooms at the bottom left, with the last robot moving to its home location at the left of the hospital corridor). Figure 4 shows three Turtlebot 3 Burger robots[7] operating within a simulated healthcare facility comprising a total of 20 rooms (with each room located between the vertical walls and either side of the long horizontal corridor).

We assume that the factors of primary importance for the case study are robots completing their mission by the defined end time, and returning to their starting location successfully for recharging, without become stranded due to battery exhaustion. These requirements are formalised in Table 1.

We apply the methodology presented in Section 2.1.2 as follows:

Step 1: Within a newly generated project, users should provide an instantiation of the SESAME Testing DSL, specifying the structure of the testing space for the example case study. The DSL metamodel is specified in our deliverable ([6]). The Exceed editor provides a convenient visual editor in order to assist the configuration of the testing process. An example of the Testing DSL model is presented in Figure 5 for the case study.

Step 2: Code generation can be used to generate metric templates automatically, within the newly generated project under the child Eclipse instance. The testing platform provides a plugin consisting of a wizard with a single page, which can be accessed by right-clicking on the user's newly generated project and selecting "Generate SESAME Code" (Figure 6a). The plugin provides an interface option to select the file containing the user's populated model (Figure 6b):

[7]https://www.turtlebot.com

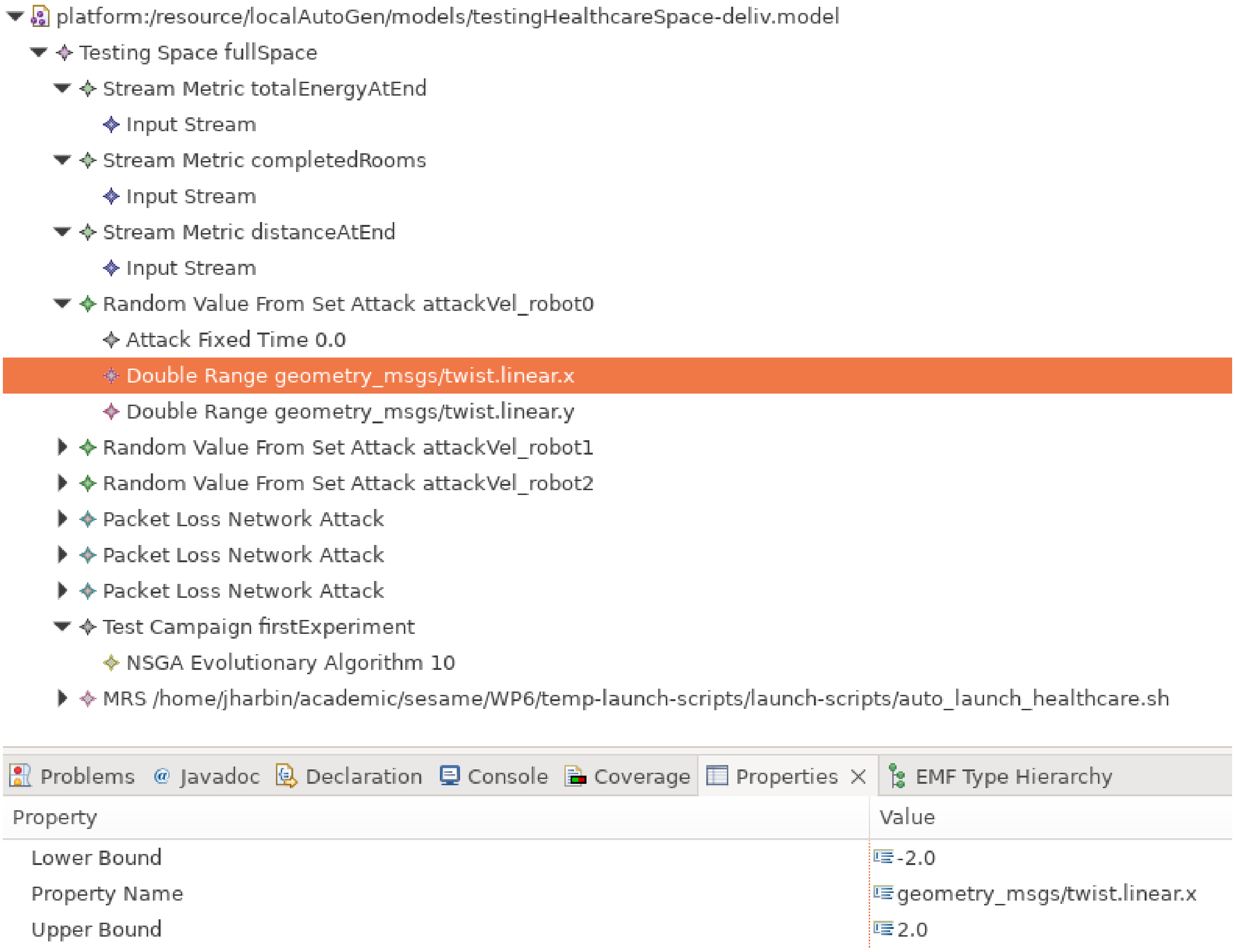


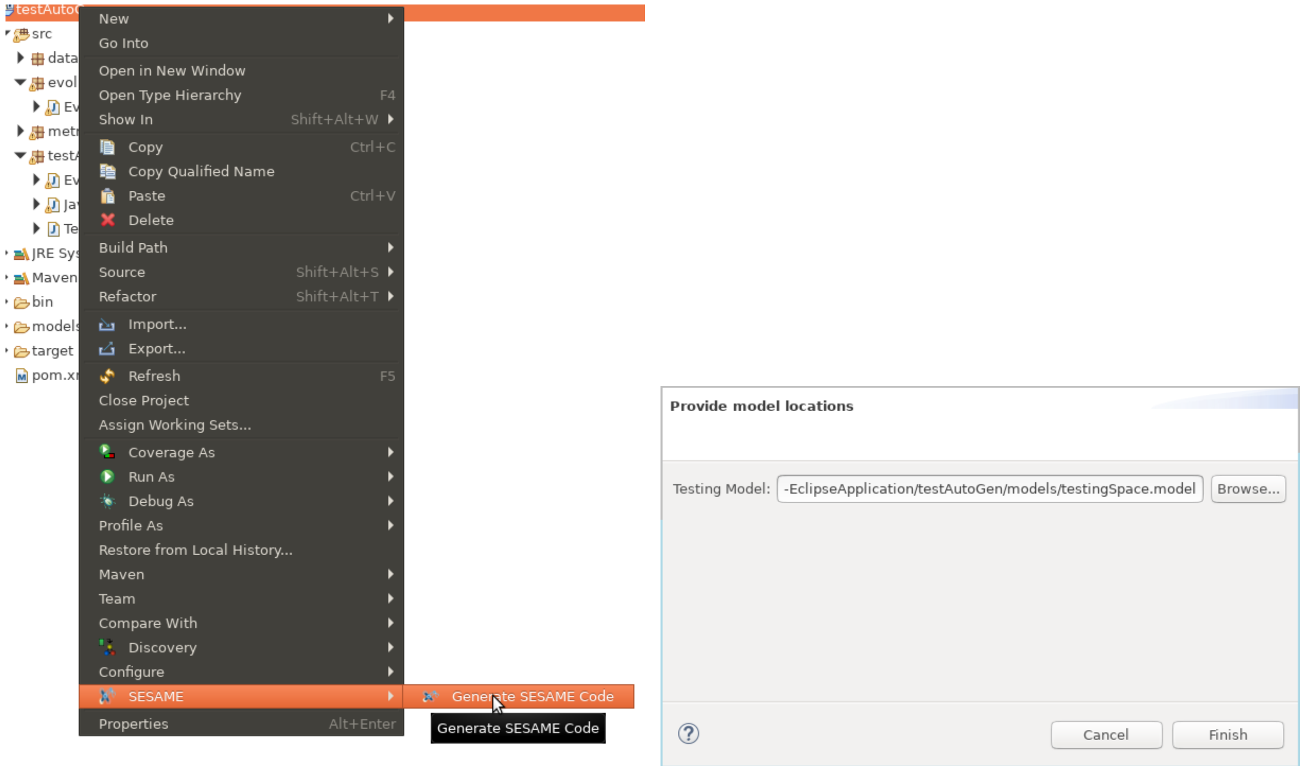
Figure 5: Testing Space Model for Simulation-Based Testing

When this button is clicked, metric templates will be generated in the package **metrics.generated**. Experiment runners will also be generated based upon the experiment name defined for the TestCampaigns in the model, e.g., 'ExptRunner_name.java'.

After this code generation for the first time, the project structure must be changed to a Maven project. Users should right-click on the project, select "Configure" / "Convert to Maven Project". Then the user should right-click and select "Maven" / "Update Project" to ensure all the dependencies are updated.

Step 3: The next step involves the user specifying scenario-specific performance metrics, in order to quantify violations of mission requirements. In order to implement these metrics, the user first needs to copy these classes from **metrics.template** into a new package **metrics.custom**. Then, it is necessary to implement the needed platform-specific metrics as Java code. To this end, users need to implement the 'processElement' stub function if the metric is connected only to a single simulator stream, and both 'processElement1' and 'processElement2' functions if the metrics are connected to both streams.

Figure 7 gives an example metric which has been implemented for tracking the number of successfully completed rooms within the healthcare case study. It listens to incoming simulator events on the simulator IN stream (receiving messages from the MRS) and reacts to messages from the MRS on any robot's topic roomCompleted, whose value contains the ID of a newly completed room. The metric code responds by incrementing its room completion counter. The total number of completed rooms at that instant is emitted as the metric's output, which can also be logged as a time series by the simulation-based testing interface.



(a) SESAME Wizard activation

(b) SESAME Wizard model file selection

Figure 6: Code generation by activating SESAME wizard

Step 4: The user should create an Eclipse Run Configuration for the class **ExptRunner_name.java** for the experiment they would like to execute, and invoke this Run Configuration in order to run the experiment. This runner will be configured with the parameters chosen from the Testing DSL. If any parameters of the experiment in the model are changed, users should rerun "Generate SESAME Code" in Step 2, to ensure this experiment runner is updated.

Before running the experiment, it is necessary to set certain paths so the dynamic code generation and experimental processes can be launched properly. Open the newly generated class in the **main** package, under the **ExptRunner_name.java** and follow the instructions within it regarding "Setting Paths". Additional information about the paths required for the experiments can be found online in the README file at at `https://github.com/sesame-project/simulationBasedTesting`.

The experiment runner will generate tests according to the strategy specified for the experiment. For example, with NSGA-II [2], it will perform multi-objective optimisation using the NSGA-II algorithm. A population of tests is generated and evaluated. Each test is evaluated by first dynamically generating a specialised test runner which acts as a middleware interfacing with the low-level MRS simulation and modifying its internal messages, using any custom supplied metric definitions provided in Step 2 to quantify the impact of the fuzzing test.

This information is communicated to the experiment runner and used to guide a multi-objective optimisation process. This process uses genetic operations such as mutation and crossover to create new tests, discarding the worst performing campaigns from the population. The metric results from the experiment are logged back to the model as 'MetricInstances' under the performed tests in the model. An example model, showing some synthetic results given as metric instances under individual tests, is presented in the file `https://github.com/sesame-project/simulationBasedTesting/blob/main/uk.ac.york.sesame.testing.architecture.testing/example-models/testingHealthcareSpace-results.model`.

```java
package metrics.custom;

import org.apache.flink.api.common.state.*;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.*;
import org.apache.flink.util.Collector;

import uk.ac.york.sesame.testing.architecture.data.EventMessage;

public class completedRoomsMetric extends ProcessFunction<EventMessage, Double> {

        private static final long serialVersionUID = 1L;
        private ValueState<Long> totalRoomsCompleted;

    public void open(Configuration parameters) throws Exception {
        totalRoomsCompleted = getRuntimeContext().
            getState(new ValueStateDescriptor<>("totalRoomsCompleted", Long.class));
    }

    public void processElement(EventMessage msg, Context ctx, Collector<Double> out) throws Exception {
        String completionTopicName = "roomCompleted";
        if (msg.getTopic().contains(completionTopicName)) {
                Object v = msg.getValue();
                totalRoomsCompleted.update(totalRoomsCompleted.value() + 1);
        }

        out.collect(Double.valueOf(totalRoomsCompleted.value()));
    }
}
```

Figure 7: An example metric implementation for the healthcare case study

2.3.1 Use Cases

The same methodology will be applied to the SESAME Use Cases. The ROS simulator interfaces used for this healthcare case study can be repurposed to operate with the Locomotec case study, since it uses a common ROS interface (rosbridge). Equally the ROS interface can be used for the KIOS use case, although it may require additional development work to interface with multiple ROS masters. We also have an in-progress interface for the TTS simulator for the KUKA use case, using the gRPC protocol. Finally, we are exploring the interfaces that need to be developed to support the communication with the simulators used by our other project partners (i.e., AVL and DKOX).

3 Test Tools for ML-Enabled Components

3.1 Overview

This section presents an overview of the main goals that led to the creation of the Machine Learning testing component of SESAME, and a brief description of the DeepKnowledge approach along with a detailed manual for its use. The information provided in this section complements deliverable D6.1 [5].

ML-enabled systems are more versatile and are capable of working in changing environments and adapting to them. For instance, vision systems used in robotics are combined with ML algorithms, Deep Neural Networks (DNNs) in particular, to improve the prediction or detection accuracy. Besides all achievements and the notable advances made recently in DNNs, possible risks due to reduced generalisation capabilities and accurate response to inputs not seen during training incur major issues that threaten the DNNs robustness.

The DeepKnowledge approach analyses how the model builds and transfers knowledge under domain shift. Through this analysis, we establish a fundamental understanding of knowledge generalisation abilities of the model at the level of individual neurons, and quantify the individual contribution of each neuron to this process.

Through additional filtering steps, DeepKnowledge identifies a set of neurons, termed transfer knowledge neurons, that are considered core units of the DNN. This set is used as a *test adequacy criterion* to determine activation values clusters that reflect the changes in neurons behaviour with respect to new inputs.

Then, we measure the diversity of the input set as the degree to which the input set covers the clusters of identified transfer knowledge neurons. In particular, given a test set Y, a coverage score is calculated to measure the systematic diversity of Y. Higher coverage means more combinations of transfer knowledge neurons clusters have been exercised. Hence, the targeted test set Y is '*knowledge-diverse*' and supports systematic testing of the target DNN model.

The following sections provide a brief technical description of the DeepKnowledge approach developed in work package WP6. Furthermore, we provide a user manual for the prototype DeepKnowledge tool. To this end, we have identified the requirements that should be satisfied for enabling the integration and possible reproduction of the experimental results carried out within Task 6.1.

3.2 DeepKnowledge

Figure 8 shows our DeepKnowledge approach for testing DNNs. The approach involves three main steps, i.e., knowledge generalisation, then using the identified neurons that are capable for transferring knowledge to specialise the test adequacy criterion, and, finally, using data augmentation techniques to synthesise new tests that increase the coverage. The application of these steps increases our confidence in the correct functionality of the DNN model and reduces the risk for erroneous behaviour due to limited testing.

Step 1: Knowledge Generalisation. This step allows to analyse the DNN model's generalisation ability, and thus to identify neurons that are core contributors to decision-making. This step encompasses the actions *knowledge abstraction analysis* and *knowledge shift/change quantification*. These sub-modules have been detailed in deliverable D6.1 [5], along with the assumptions and goals underlying model generalisation.

At this step, we use the activation maximisation methodology implemented using the DNN framework Keras (v2.2.2) with Tensorflow (v2.6) as its backend. This methodology is used to quantify both the knowledge abstraction and shift at neuron level.

The output of this step is a probability distribution over the In-Domain and Out-Of-Domain datasets. Please refer to deliverable D6.1 for more details.

Step 2: Coverage Estimation. In this step, two main actions are performed:

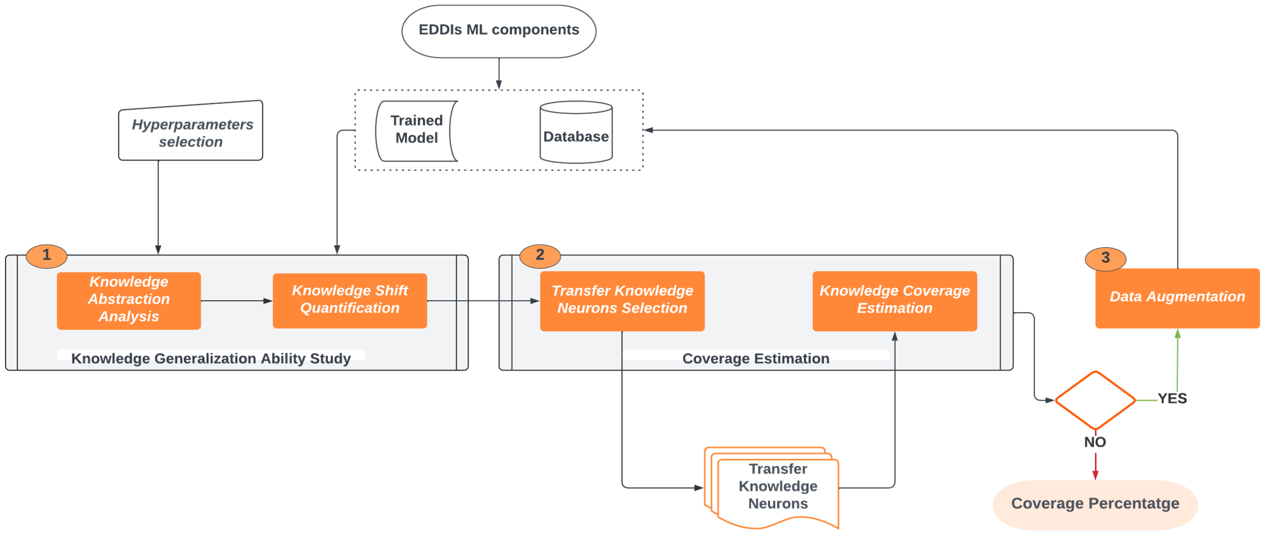


Figure 8: Overview of DEEPKNOWLEDGE testing methodology

(i) *Transfer knowledge neurons selection*, where the main computational units are identified, i.e., neurons that are responsible for knowledge generalisation. Our approach allows to identify these components, so called transfer knowledge neurons, by applying a symmetric divergence metric. The divergence metric deployed in this first experimental study is Hellinger distance [1] (implemented using Python libraries). Our approach is generic and can support different divergence metrics, including the well-known Wasserstein distance [4], for example. Please note that other distance measures can be implemented with the same Python libraries.

(ii) *Knowledge coverage estimation*, where we mainly quantify the number of cluster combinations covered by the given test set. This quantification provides an estimate of how knowledge diverge the test set is. To support this estimation, we leverage the *DeepImportance* work [3] [8] and make small adaptations that are needed to support our approach.

Step 3: The final step in this workflow is data augmentation. The estimation of coverage for a given test set will allow us to assess the correctness of the model. Depending on the final coverage percentage, we employ a DEEPKNOWLEDGE-based model augmentation strategy. This strategy aims to augment the training set of a DNN using a set of selected inputs generated using DEEPKNOWLEDGE's test adequacy criterion, i.e., based on the identification of transfer knowledge neurons (cf. deliverable D6.1 for more details). Then, we deploy these inputs to retrain the DNN and evaluate if they help improving the DNN's accuracy.

3.3 User Guide

All experiments in our study were conducted on a high-performance computer running a cluster GPU with NVIDIA 510.39. We implemented the proposed DEEPKNOWLEDGE approach and the knowledge transfer neuron selection method based on the state-of-the-art framework open-source DNN framework Keras (v2.2.2) with Tensorflow (v2.6) backend.

[8] https://www.github.com/DeepImportance/deepimportance_code_release

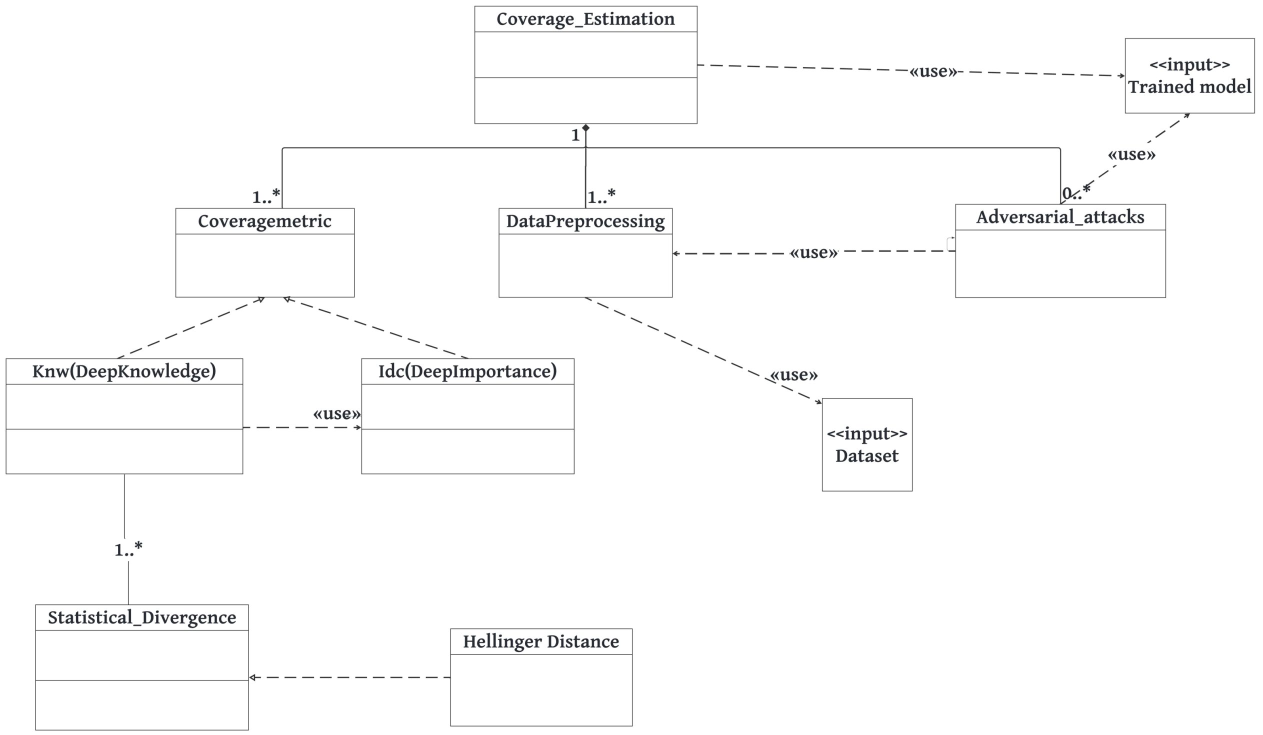


Figure 9: DEEPKNOWLEDGE component diagram

DEEPKNOWLEDGE is developed using the Python programming language. All the Python scripts enabling the use and integration of this DNN testing tool are available on Github[9] as a set of scripts linked as shown in Figure 9.

The code is produced as a set of packages and classes. The main script is *Coverage_Estimation.py*. The package *Coverages* includes our coverage metric of DEEPKNOWLEDGE under the script *knw.py* and the DeepImportance under the script *idc.py*. Other state-of-the-art coverage metrics are also available under that package.

The dataset class allow to load and preprocess different datasets for the purpose of testing the model and quantifying its knowledge. It can also be deployed to generate adversarial examples by using the *adversarial technique* class. The Hellinger distance class is available through the *hellinger_distance.py* script.

3.3.1 SETTING UP THE ENVIRONMENT: REQUIRED LIBRARIES

The complete installation guide is presented as a README guide within the Github repository at `https://github.com/sesame-project/ML_Testing`. After completing the installation process indicated there, please continue with the user guide below.

3.3.2 RUNNING THE DEEPKNOWLEDGE TOOL

The DEEPKNOWLEDGE tool can be run as mentioned using shell commands. The user can execute the commands by running the shell script *Coverage_Estimation.py*.

[9] https://github.com/sesame-project/ML_Testing

- model => The name of the Keras model file. Note the architecture file (i.e., JSON) and the weights file should be saved separately as an .h5 file. If the model is trained and saved into a file, it needs to be in the (.hdf5) format. You need also to save all the model under the same folder *'Networks'*.

- dataset = Name of the dataset to be used. Current implementation supports 'MNIST', 'Cifar10' and 'Cifar100', 'fashion MNIST' and 'SVHN'. The code is configurable and adding another is possible through a loading function in the script 'Dataprocessing.py'.

- approach = The approach for coverage estimation. This includes DEEPKNOWLEDGE noted as 'knw', and other baselines implemented for our empirical study. Current implementation supports DeepImportance 'idc','nc','kmnc','nbc','snac','tknc','ssc', 'lsa', and 'dsa'.

- neurons = The number of neurons used in 'knw'

- threshold = a threshold value used to consider if a neuron is a transfer knowledge neuron or not for 'knw'.

- advtype = The name of the adversarial attack to be applied. This implementation supports 'mim', 'bim', 'cw', and 'fgsm' techniques.

- class = selected classes. Note this argument is used for approaches like 'kmnc','nbc','snac' and has no use for 'knw' context. This argument takes a number between 0 and 9 for mist or cifar10 and svhn.

- layer = The subject layer's index for approaches including 'idc','tknc', 'lsa'. Note that only trainable layers can be selected.

- logfile = The name of the file that the results to be saved.

- repeat = Obtained results are added to repeat the experiments to reduce the randomness effect.

Figure 10: Parameters for configuring DEEPKNOWLEDGE

Command Line Settings The script expects arguments to alter their behaviour. These arguments represent the hyper-parameters that can be manually specified. We provide a detailed description for each of these parameters in Figure 10.

Example Command

```
python Coverage_Estimation.py –model [path_to_keras_model_file] –dataset svhn –approach knw –threshold 0.5 –logfile [path_to_log_file]
```

The user also can run the shell script with the default settings and without specifying any argument as we show in the following example.

3.4 EXAMPLE

In order to illustrate the DEEPKNOWLEDGE prototype tool, we represent an example on a computer vision task. The task is image classification with the DNN model Resnet-18.

- model = ResNet18

- dataset = svhn

- approach = knw

- neurons = 10

- threshold = 0.5

- advtype = None

- class = None

Note that the DEEPKNOWLEDGE approach does quantify the knowledge change over all the trainable layers on a DNN - layer = ALL

- logfile = result.log

- repeat = 0

Figure 11: DEEPKNOWLEDGE default settings

ResNet-18 is a convolutional neural network that is 18 layers deep. The model is composed of several residual blocks and skip connection is used in the residual block, which solved the problem of performance degradation of deep convolution neural networks. Besides its notable performance, testing this neural network architecture against the publicly-available dataset SVHN will increase our confidence for the robustness of ResNet18 in image classification. The SVHN dataset is a real-world image dataset for object recognition obtained from house numbers in Google Street View images.

We apply the methodology presented in Section 3.2 by running the shell script with the default settings (Figure 11).

1 - Start by activating the virtual environment created to run DEEPKNOWLEDGE, i.e., virtual env, called SESAME in this case as shown in Figure 12.

  - We recommend creating a separate virtual environment to keep dependencies required by the DEEPKNOWLEDGEproject separate.
  - To create a Python virtual environment you can simply run the following command on Linux: `python3 -m venv [name_of_virtual_environment]`
  - To activate the virtual environment run: `source [name_of_virtual_environment]/bin/activate`

2 - Go to the repository folder EU_SESAME in a terminal

3.1 - Run the DEEPKNOWLEDGE script with default settings using `python3.8 Coverage_Estimation.py`

3.2 - or Run the DEEPKNOWLEDGE script with customised parameters as shown in Figure 14.

4 - On the terminal you will get the coverage results as shown in Figure 15.

```
[redacted]$ source SESAME/bin/activate
(SESAME) [redacted]$
```

Activate a virtual environment

Figure 12: Shell command to activate a virtual environment.

Python script command

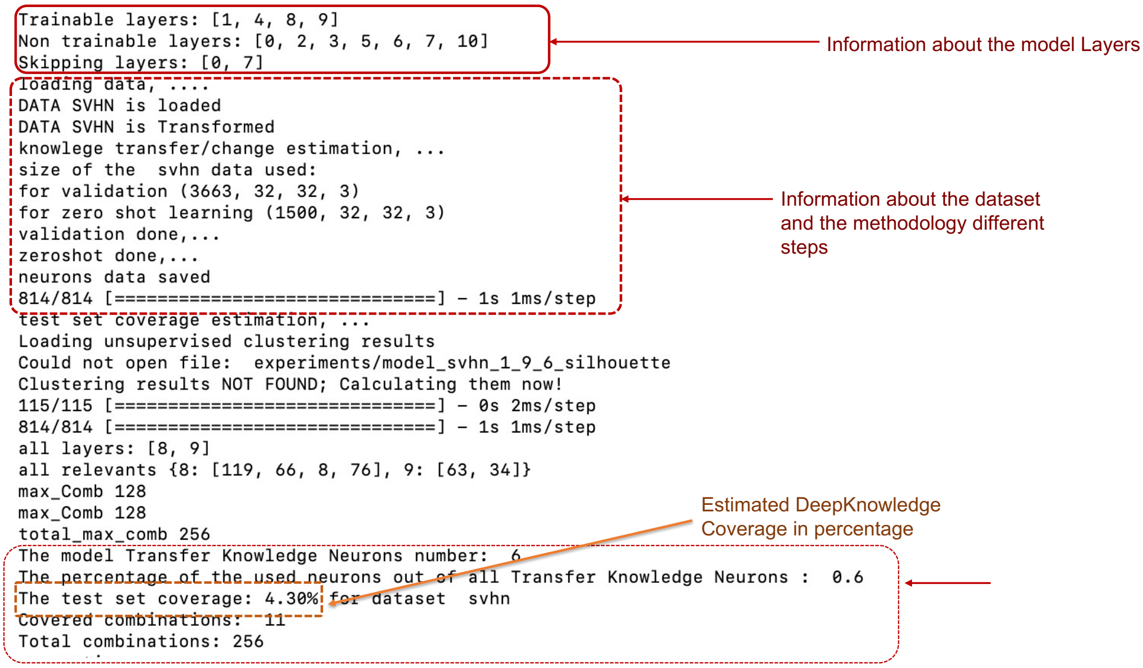
```
(SESAME) [redacted] python3.8 Coverage_Estimation.py
11.2
2022-06-21 22:21:20.086121: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1),
 NUMA node zero
2022-06-21 22:21:20.107808: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1),
 NUMA node zero
2022-06-21 22:21:20.107946: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1),
 NUMA node zero
2022-06-21 22:21:20.108247: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-06-21 22:21:20.108904: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1),
 NUMA node zero
2022-06-21 22:21:20.109053: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1),
 NUMA node zero
2022-06-21 22:21:20.109181: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1),
 NUMA node zero
2022-06-21 22:21:20.393715: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1),
 NUMA node zero
2022-06-21 22:21:20.393876: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1),
 NUMA node zero
2022-06-21 22:21:20.393983: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1),
 NUMA node zero
2022-06-21 22:21:20.394072: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1532] Created device /job:localhost/replica:0/task:0/device:GPU:0 with
X 3080, pci bus id: 0000:01:00.0, compute capability: 8.6
exeception
Trainable layers: [1, 4, 8, 9]
Non trainable layers: [0, 2, 3, 5, 6, 7, 10]
Skipping layers: [0, 7]
loading data, ....
```

Figure 13: DEEPKNOWLEDGE example shell command line to run DEEPKNOWLEDGE tool-supported technique.

Python script command

```
(SESAME) [redacted] cd EU_SESAME
(SESAME) [redacted]$ python3.8 Coverage_Estimation.py -model Networks/svhn -dataset svhn -approach knw -threshold 0.5 -logfile svhnresults.log
11.2
2022-06-29 13:24:15.881927: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1), but there must be at
 NUMA node zero
2022-06-29 13:24:15.903807: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1), but there must be at
 NUMA node zero
2022-06-29 13:24:15.903938: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1), but there must be at
 NUMA node zero
2022-06-29 13:24:15.904234: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to us
in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-06-29 13:24:15.904805: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1), but there must be at
 NUMA node zero
2022-06-29 13:24:15.904980: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1), but there must be at
 NUMA node zero
2022-06-29 13:24:15.905155: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:975] successful NUMA node read from SysFS had negative value (-1), but there must be at
 NUMA node zero
```

Figure 14: Shell command to run DEEPKNOWLEDGE tool-supported technique with specific hyper parameters.



```
Trainable layers: [1, 4, 8, 9]
Non trainable layers: [0, 2, 3, 5, 6, 7, 10]
Skipping layers: [0, 7]
loading data, ....
DATA SVHN is loaded
DATA SVHN is Transformed
knowlege transfer/change estimation, ...
size of the  svhn data used:
for validation (3663, 32, 32, 3)
for zero shot learning (1500, 32, 32, 3)
validation done,...
zeroshot done,...
neurons data saved
814/814 [==============================] - 1s 1ms/step
test set coverage estimation, ...
Loading unsupervised clustering results
Could not open file:  experiments/model_svhn_1_9_6_silhouette
Clustering results NOT FOUND; Calculating them now!
115/115 [==============================] - 0s 2ms/step
814/814 [==============================] - 1s 1ms/step
all layers: [8, 9]
all relevants {8: [119, 66, 8, 76], 9: [63, 34]}
max_Comb 128
max_Comb 128
total_max_comb 256
The model Transfer Knowledge Neurons number:  6
The percentage of the used neurons out of all Transfer Knowledge Neurons :  0.6
The test set coverage: 4.30% for dataset  svhn
Covered combinations:  11
Total combinations: 256
```


Figure 15: Output results of DEEPKNOWLEDGE shell command scripts – testing ResNet18 model against the SVHN dataset.

The same methodology will be applied to the SESAME uses cases. The shell scripts used for this example can be repurposed for Locomotec's case study with few adaptations to support object detection. In particular, we need to adapt the 'load_model' function in *utils.py* script to load DNNs from the family used to train Locomotec's models.

Similarly, some small changes will be needed in the *Dataprocessing.py* script to enable loading and preprocessing video datasets for object detection such as the COCO dataset COCO dataset used by our project partner Locomotec.

4 Conclusion

In this document, we presented the architectural specification and implementation details of the ML Testing and Simulation-Based Testing components of SESAME developed in Tasks 6.1 and 6.2. This document complement deliverables D6.1 and D6.2, providing also a user guide to facilitate the use of the developed software.

By following the agreed architectural conventions specified for the integrated platform of SESAME and the partners requirements from deliverable D1.1, we identified the main models, data types, and processes developed for the assurance techniques of ML testing and simulation-based testing.

References

[1] Rajendra Bhatia, Stephane Gaubert, and Tanvi Jain. Matrix versions of the hellinger distance. *Letters in Mathematical Physics*, 109(8):1777–1804, 2019.

[2] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[3] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. Importance-driven deep learning system testing. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 702–713. IEEE, 2020.

[4] Ludger Rüschendorf. The wasserstein distance and approximation theorems. *Probability Theory and Related Fields*, 70(1):117–129, 1985.

[5] SESAME Project Partners. SESAME: Deliverable D6.1: Assurance of Data-Driven and Learning Components of EDDIs. Technical report, University of York, Jun 2022.

[6] SESAME Project Partners. SESAME: Deliverable D6.2: Simulation-Based Testing Methodology for EDDIs. Technical report, University of York, Jun 2022.