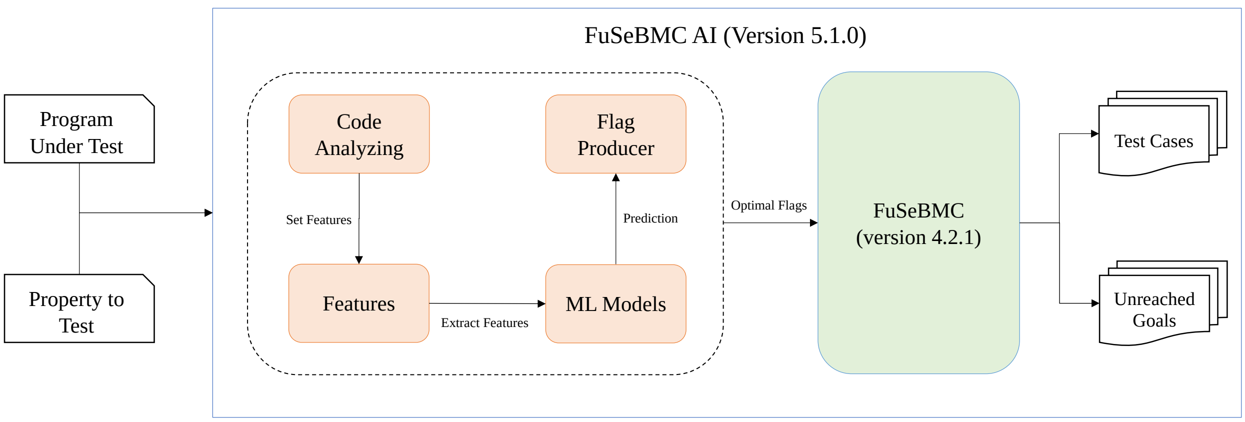
FuSeBMC AI: Acceleration of Hybrid Approach through Machine Learning

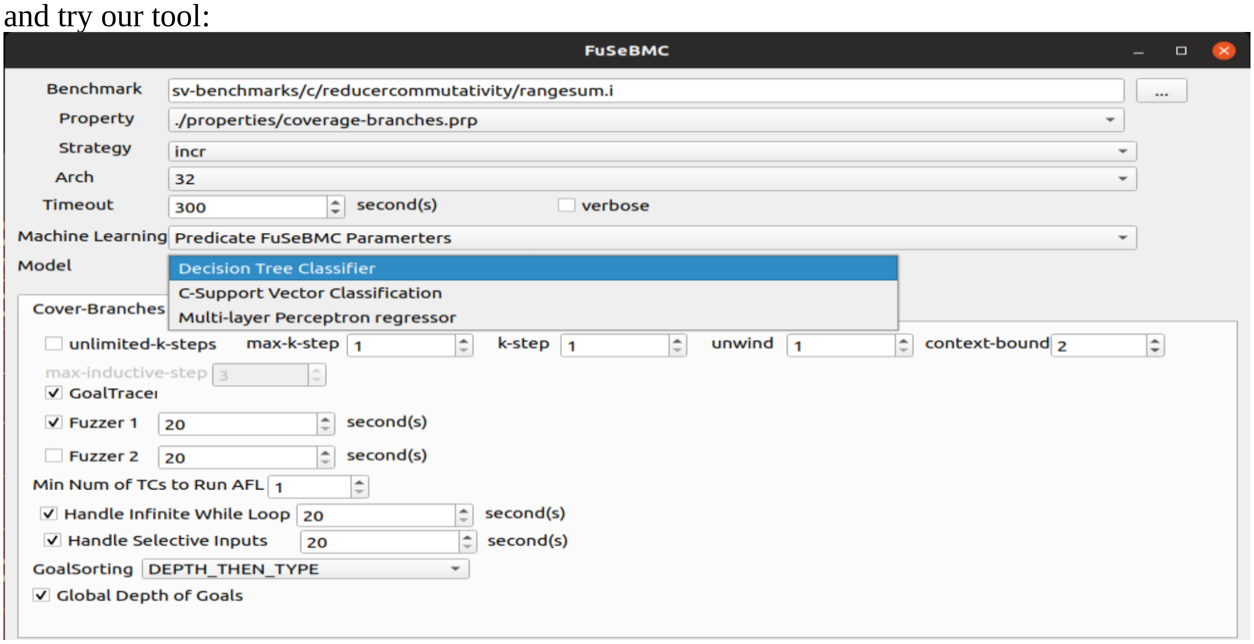
FuSeBMC-AI is a test generation tool grounded in machine learning techniques. FuSeBMC-AI extracts various features from the program under examination and employs a support vector and neural network to predict a hybrid approach's optimal configuration. Our current research specializes in Software Testing and utilizes BMC and Fuzzing as back-end verification engines. FuSeBMC-AI exhibits enhancements in some subcategories in Test-Comp 2024, when compared to the default configuration of FuSeBMC, concurrently achieving a 3% reduction in resource utilization as shown in the results of Test-Comp 2024 experiments.

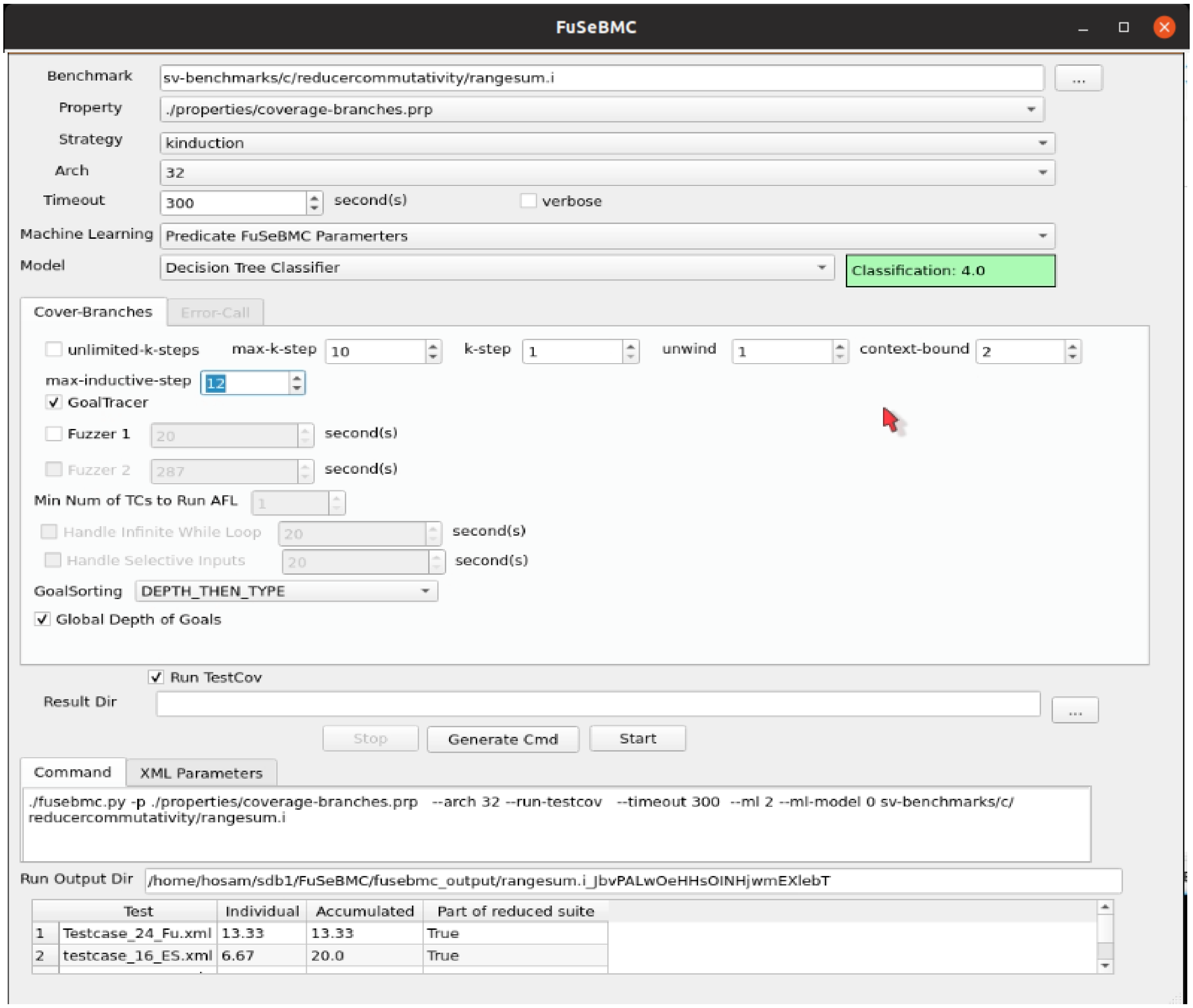
Framework of FuSeBMC-AI:



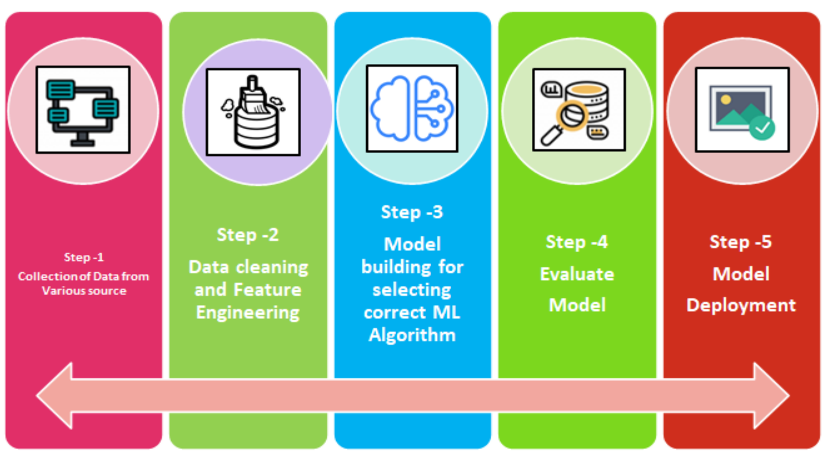
GUI of FuSeBMC-AI:

We have developed an interface for FuSeBMC-AI so we can make it easy for the user to conduct and try our tool:





FuSeBMC-AI applies the concept of machine language operations. It begins with collecting data, cleaning it, applying the model, evaluation, and then development as shown below:



In the **data collection** phase, we used Test-Comp's Benchmark for categories Cover-Error and Cover-Branches So that we ensure diversity in the programs and scenarios that we may face. We chose the train programs as follows:

Cover-Branches

Category	Total	chosen	Percent
SoftwareSystems-DeviceDriversLinux64-ReachSafety	290	13	4%
SoftwareSystems-SQLite-MemSafety.	1	0	0%
ReachSafety-Arrays	400	10	2%
ReachSafety-Floats	226	10	4%
ReachSafety-Combinations	671	13	2%
ReachSafety-ProductLines	263	7	3%
ReachSafety-Heap	143	9	6%
ReachSafety-BitVectors	62	4	6%
SoftwareSystems-BusyBox-MemSafety	75	6	8%
ReachSafety-Recursive	53	4	8%
Termination-MainHeap	231	11	5%
ReachSafety-Sequentialized	103	8	8%
ReachSafety-ControlFlow	67	5	7%
ReachSafety-ECA	29	6	21%
ReachSafety-XCSP	119	5	4%
TOTAL	**2733**	**111**	**4%**

Cover-Error

Category	Total	chosen	Percent
ReachSafety-ECA	18	2	11%
ReachSafety-ProductLines	169	16	9%
ReachSafety-Arrays	100	7	7%
ReachSafety-Recursive	20	5	25%
SoftwareSystems-BusyBox-MemSafety	13	2	15%
ReachSafety-Sequentialized	110	7	9%
ReachSafety-XCSP	59	7	12%
ReachSafety-BitVectors	10	2	20%
ReachSafety-ControlFlow	32	5	16%
SoftwareSystems-DeviceDriversLinux64-ReachSafety	2	1	50%
ReachSafety-Heap	56	6	11%
ReachSafety-Floats	33	4	12%
TOTAL	**619**	**67**	**11%**

Program Features:

FuSeBMC-AI focuses on discerning the features whose values could impact the efficacy and limitations of the engine's performance. This emphasis arose from recognizing that certain programs necessitate specific values for effective handling, particularly those involving arrays and extensive loops. After conducting initial experiments, it was determined that the most influential features outlined in tables below besides examples of flags that might pass to FuSeBMC-AI's engines.

	FEATURE	VALUE
FOR	forCount	The total number of For loops in the source code
	forMaxDepth	The maximum depth of for loops
	forDepthAvg	forCount/sum(depths of For Loops)
WHILE	whileCount	
	whileMaxDepth	
	whileDepthAvg	
	whileInfiniteCount	Number of infinite while loops {while(1){}}

	WhileInfiniteWithNonDetCallCount	While(1){
		x = __VERIFIER_nondet_int();
		}

Do {} while()	doCount	
	doMaxDepth	
	doDepthAvg	
	doInfiniteCount	Do {} while(1)
IF	ifCount	
	ifMaxDepth	
	ifDepthAvg	

	nestedIfCount	If()
		{
		if()
		}

ELSE	elseCount	
	elseDepthAvg	
Verifier input functions	NonDetCallCount	/FuSeBMC/FuSeBMC_FuzzerLib/src/FuSeBMC_FuzzerLib.c
	NonDetCallDepthAvg	
	HasNonDetCallInLoop	0 or 1
Concurrency	hasConcurrency	0 or 1

Parameters combinations

The model must be trained on all possible values of the parameters and the results of their implementation in order for it to be able to give us the best parameters for a particular program.

Cover-Branches:

Parameter	**Values**
Strategy	incr, kinduction
Solver	boolector, z3
Encoding	floatbv, fixedbv
KStep	[1,2,3]
ContextBound	[2,4]
Unwind	[10,-1] #-1 default
Fuzz1Enabled	[0,1]
Fuzz1Time	[25,83,188] for 250 seconds (300 - 50) # 10%, 33.3%, 75%
TOTAL	**2*2*2*3*2*2*4=384**

Cover-Error:

Parameter	**Values**
Strategy	incr, kinduction
Solver	boolector, z3
Encoding	floatbv, fixedbv
KStep	[1,2,3]
ContextBound	[2,4]
Fuzz1Enabled	[0,1]
Fuzz1Time	[25,83,188] for 250 seconds (300 - 50) # 10%, 33.3%, 75%
TOTAL	**2*2*2*3*2*4=192**

ML Models:

The primary focus involved four models, namely the Decision Tree Classification (DTC), Support Vector Classification (SVC), Neural Network Regression (ANN), and a multi model (DTC then SVC then NNR)). The training phase was executed, followed by testing using the aforementioned benchmarks. The four models underwent supervised and guided training, you say supervised learning; where are the labels (ie ground truth) come from? ensuring a balanced approach to mitigate repetition during the training phase. The training process involved teaching the models to predict optimal parameters for FuSeBMC-AI's engines, thereby assisting these engines in determining the most suitable parameter values for each category of programs. The classification of outputs was dedicated to facilitating model training (see Fig 3). The classification process involved categorizing ”Cover-Error” and ”Cover-Branches.” This categorization was based on the extent of coverage or error detection and the corresponding time duration. Comprehensive testing of approximately 384 parameter values (for each program) was conducted.

Models Training:
As is known, in machine learning, the model is trained on Features and Target. The Feature is the Source code Features + Coresponding Parameter combination, and the Target is the class or score that the file obtained as a result of executing it with the specified parameters.

LABLING and SCORING

```
/FuSeBMC/FeatureFromXMLResultExtractor.py method setScoreAndClass
classes :6 classes,  0,1,2,3,4,5; best class is 0 ; worsrt is 5
scores :
        coverage-error-call : 0 or value between [50,100]
        coverage-branches : from 0 to 101

SCORING in Coverage-error-call
IF originalScore = 0 THEN
        ourScore = 0
        ourCalss = 5
ELSE IF originalScore = 1 THEN
        CONST resultTimeLimit = 300
        restTime = resultTimeLimit -  execTime
        IF restTime <= 5 THEN restTime = 0
        restTimeRatio = restTime / resultTimeLimit
        ourScore = 50 + (50 * restTimeRatio)
        IF restTimeRatio >= 0.8 THEN
                ourCalss = 0
        ELSE IF restTimeRatio >= 0.6 THEN
                ourCalss = 1
        ELSE IF restTimeRatio >= 0.4 THEN
                ourCalss = 2
        ELSE IF restTimeRatio >= 0.2 THEN
                ourCalss = 3
        ELSE IF restTimeRatio >= 0.0 THEN
                ourCalss= 4
        END
END
```

Example:

restTime = 300 - 238.12=61.88
restTimeRatio = 61.88 / 300 = 0.206
ourCalss = 3
*ourScore = 50 +(50 * 0.206) = 60.3*

SCORING in Coverage-branches
*ourScore = origScore * 100*
IF ourScore = 100 THEN ourScore += restTimeRatio
IF origScore >= 0.85 THEN ourCalss = 0
ELSE IF origScore >= 0.68 THEN ourCalss = 1
ELSE IF origScore >= 0.51 THEN ourCalss = 2
ELSE IF origScore >= 0.34 THEN ourCalss = 3
ELSE IF origScore >= 0.17 THEN ourCalss = 4
ELSE IF origScore >= 0.0 THEN ourCalss = 5
END

Testing Result (Cover-Error)	**Class**
detect bug & IF restTimeRatio >= 0.8	0
detect bug & ELSE IF restTimeRatio >= 0.6	1
detect bug & ELSE IF restTimeRatio >= 0.4	2
detect bug & ELSE IF restTimeRatio >= 0.2	3
detect bug & ELSE IF restTimeRatio >= 0.0	4
Unknown	5

Coverage Result (Cover-Branches)	**Class**
score coverage >= 0.85	0
score coverage >= 0.68	1
score coverage >= 0.51	2
score coverage >= 0.34	3
score coverage >= 0.17	4
score coverage >= 0.0	5

After training the model, it is able to give us the parameter combinations that give the best class or the best score for a specific program.

Evaluation phase:

We selected 10% of each category in the table above automatically to conduct our experiments. It also contains different programs such as result and execution time.

- Experiments:
  We conducted experiments on benchmarks taken from the 2023 SV-COMP in different properties (coverage-branches, coverage-error-call, no-overflow, termination, unreach-call and valid-memsafety) , which contain various open-source applications, e.g., bftpd, which is an FTP server for Unix systems.

Property	FuSeBMC Plain / Time(s)	DTC/Time(s)	SVC/Time(s)	NNR/Time(s)	# of Programs
coverage-error-call	52 in 4550s	50 in 2310s	54 in 2980s	52 in 3150s	60
coverage-branches	49.9 in 25000s	43.6 in 17000s	43.1 in 17000s	43.0 in 14400s	83
no-overflow	25 in 9670s	25 in 2350s	25 in 2300s	25 in 8840s	63
termination	136 in 42000s	142 in 8150s	142 in 8140s	146 in 34200s	229
unreach-call	328 in 153000s	41 in 67800s	41 in 67800s	254 in 65700s	432
valid-memsafety	404 in 15300s	176 in 5610s	176 in 5620s	312 in 11200s	344

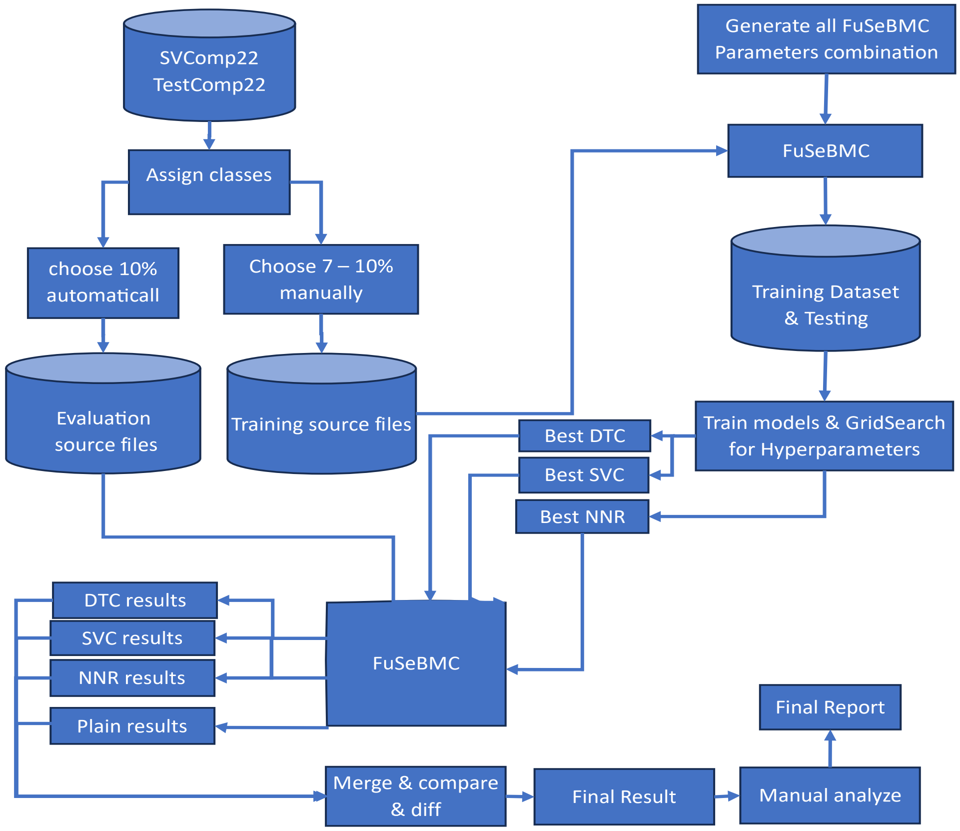
For each of the training benchmarks, we run FuSeBMC-AI for 300s time limit, 8000 MB memory limit and 1 CPU core limit with 384 different combinations of flags.

You can find the results in the link below:
https://drive.google.com/drive/folders/1yjb7duAj-Cyi2cHZbnLvB5dAHrvSHks5?usp=sharing

PsudoCode:

```
GLOBAL VARIABLES: DSFile, modelFile :string

FUNCTION getParamsCombinations()
BEGIN
        p1Vals = [p1v_1,p1v_2,….p1v_m]
        p2Vals = [p2v_1,p2v_2,….p2v_n]
        …….
        pnVals = [pnv_1,pnv_2,….pnv_o]
        all = generate all combinations from:p1Vals, p2Vals, …., pnVals
        return all
END

FUNCTION getClassOurScore (origScore, runTime)
BEGIN
        IF origScore > 80 && origScore <= 100 THEN
                oClass = 0
        ELSE IF origScore > 65 && origScore <= 80 THEN
                oClass = 1
        ELSE IF ….

        oScore = origScore + score of runTime
        return  oClass, oScore
END

FUNCTION extractFeatureFromFile (file : string)
BEGIN
        run FuSeBMC_instrument –extract-features
        return [ifCount, ForCount, …..]
END

FUNCTION runFuSeBMC (file:str, lsParams:list):
BEGIN
        run FuSeBMC with lsParams and file
        return origScore, runTime
END

FUNCTION generateTrainingData ()
BEGIN
        lsFiles = 5% until 10% from testcom22 and svcomp22
        lsAllParams = getParamsCombinations()
        FOR oneParams in  lsAllParams:
        BEGIN
                FOR fil in  lsFiles:
                BEGIN
                        srcFeatures = extractFeatureFromFile(fil)
                        origScore, runTime = runFuSeBMC(fil, oneParams)
                        ourClass, ourScore = getClassOurScore(origScore, runTime)
                        append [fil] + oneParams + srcFeatures + [ourClass , ourScore] to DSFile
                END
        END
        choose some items from DSFile as TestingData (copy randomly some files, must apply splitting 80/20 or 70/30)
END

FUNCTION trainModel()
BEGIN
        load data from DSFile
        if we do classification then drop columns ‘file’, ‘ourScore’
        if we do regression then drop columns ‘file’, ‘ourClass’
        train the model using loaded data.
        Dump the model in file ‘modelFile’
END
```

Diagram:



Tool Setup and Configuration:

When running FuSeBMC-AI, the user is required to set the architecture with -a, the property file path with -p, the competition strategy with -s, and the benchmark path, as:

fusebmc.py [-a {32, 64}] [-p PROPERTY FILE]
[-s {kinduction, falsi, incr, fixed}]
[BENCHMARK PATH]

Where -a sets the architecture to 32 or 64, -p sets the property file to PROPERTY - FILE, where it has a list of all the properties to be tested. -s sets the BMC strategy to one of the listed strategies {kinduction, falsi, incr, fixed}. The Benchexec tool info module is fusebmc.py and the benchmark definition file is FuSeBMC.xml.

Software Project:

FuSeBMC-AI is implemented using C++, and it is publicly available under the terms of the MIT License at GitHub1. The repository includes the latest version of FuSeBMC AI (version 5.1.0). FuSeBMC AI dependencies and instructions for building from source code are all listed in the README.md file. Test-Comp 2024 provides the script, benchmarks, and FuSeBMC AI binary to reproduce the competition's results.