Discovery of Endianness and Instruction Size Characteristics in Binary Programs from Unknown Instruction Set Architectures

Joachim Andreassen[1][0009−0005−0398−3115] and Donn Morrison[2][0009−0001−6072−4081]

[1] Norwegian University of Science and Technology
Trondheim, Norway
joachan@stud.ntnu.no
[2] Norwegian University of Science and Technology
Trondheim, Norway
donn.morrison@ntnu.no

Abstract. We study the problem of streamlining reverse engineering (RE) of binary programs from unknown instruction set architectures (ISA). We focus on two fundamental ISA characteristics to beginning the RE process: identification of endianness and whether the instruction width is a fixed or variable. For ISAs with a fixed instruction width, we also present methods for estimating the width. In addition to advancing research in software RE, our work can also be seen as a first step in hardware reverse engineering, because endianness and instruction format describe intrinsic characteristics of the underlying ISA.
We detail our efforts at feature engineering and perform experiments using a variety of machine learning models on two datasets of architectures using Leave-One-Group-Out-Cross-Validation to simulate conditions where the tested ISA is unknown during model training. We use bigram-based features for endianness detection and the autocorrelation function, commonly used in signal processing applications, for differentiation between fixed- and variable-width instruction sizes. A collection of classifiers from the machine learning library `scikit-learn` are used in the experiments to research these features. Initial results are promising, with accuracy of endianness detection at 99.4%, fixed- versus variable-width instruction size at 86.0%, and detection of fixed instruction sizes at 88.0%.

Keywords: Reverse engineering · Unknown Instruction Set Architecture · Machine learning · Signal processing.

1 Introduction

The emergence of IoT devices has increased the importance of understanding the workings of compiled binary files through reverse engineering (RE). Reverse engineering has applications in vulnerability research, extending support of legacy

software and hardware, binary patching and translation, and digital forensics [5].

Identifying the targeted binary file's instruction set architecture (ISA) is an essential first step in reverse engineering because it permits the reverse engineer to apply an appropriate disassembler to translate machine readable instructions into an assembly representation, and subsequently apply a decompiler that can yield high-level source code. Previous research has focused on this process by providing methods that classify ISAs reliably from a set of known ISAs [2, 3, 5, 6]. However, identifying the ISA from binary files with an unknown or undocumented ISA has not been thoroughly explored previously. Proprietary ISAs with unavailable documentation and ISAs for custom virtual machines are common examples in this group of ISAs.

This main aim of this work is to discover fundamental ISA characteristics from binary files where the ISA specification is either unknown, proprietary, or undocumented. This knowledge can be used to advance the reverse engineering process and generate documentation for unknown ISAs. As the understanding of the ISA becomes clearer, high-level program behavior such as control flow and call graph structure can be discovered [7].

We study the following research questions:

RQ1: Can machine learning be used to detect intrinsic characteristics of unknown ISAs from binary programs?
RQ2: Do the proposed approaches lead to reliable detection of ISA characteristics across a wide range of ISAs?

The main contribution of this paper is methods for discovering the following ISA characteristics:

1. endianness of the ISA,
2. fixed- versus variable-width instruction format of the ISA, and
3. for fixed instruction width ISAs, an estimation of the width

The paper is structured as follows: Section 2 presents background and related work. Section 3 presents our methodology. Section 4 presents the experimental setup and results. The results are then discussed in Section 5. A conclusion of the paper is provided in Section 6.

2 Background

A binary executable file consists of a series of executable instructions in a format understood by the CPU. When run by the CPU, a set of actions, primarily defined by a programmer, is executed. Binary executable files are generated from high-level code by a compiler that structures data into headers and segments. The headers contain information about the binary's properties and organization. The two segments central in this paper are the code and data segments. These

include the executable code and global or static variables, respectively. The code segment is particularly central, as it contains a series of instructions from which the ISA characteristics of interest are detectable [9].

The instruction set architecture (ISA) is an abstraction that specifies how CPU executes the instructions of binary programs. The specification of an ISA includes features such as endianness, instruction encoding and format, the number of physical registers, etc. [1].

Endianness describes how multi-byte values and memory addresses are ordered. This paper focuses on the two most common endianness encodings: big and little. For big endianness, the most significant bit is stored in the lowest address and the least significant bit in the greatest address. The opposite is true for little endianness.

Instruction size is the number of consecutive bits that define an instruction. The instruction size can be fixed or variable for a given ISA, and in some cases an ISA can support both fixed and variable formats (an example is the RISC-V ISA, which supports 32-bit width instructions with 16-bit extensions). Binary programs compiled for ISAs with a fixed instruction width contain only instructions of the specified size [8].

2.1 Related Work

Existing research in the field of ISA detection focuses on detecting the ISA from a predefined set of architectures. This differs from this paper's goal, which focuses on unknown ISAs that cannot be classified from a set of architectures, aiming to streamline reverse engineering of such architectures. However, these differences do not eliminate the relevance of previous research, where various approaches are usable in this research.

A paper by Kairajärvi et al. [4] contributes in two ways. First, it provides the comprehensive *IsaDetect* dataset with 66685 binary programs from 23 different architectures scraped from the Debian repositories. This is a balanced dataset, as each architecture contains a similar-sized sample. The dataset contains binary programs exclusively of either complete or code-only binary programs. These two versions of the dataset are in this paper referred to as *IsaDetectFull* and *IsaDetectCode*, respectively.

Secondly, Kairajärvi et al. [4] explores state-of-the-art methods for detecting ISAs using the IsaDetect datasets. This involves a series of features used with machine learning. With this method, the paper can classify the ISA from a predefined set with an accuracy of 98% with models trained and tested on the IsaDetectCode dataset.

The CpuRec project by Granboulan [3] contribute a dataset and command-line tool for classifying ISAs. The tool computes the Kullback–Leibler divergence between a given binary executable file (the query) and each binary executable file in the dataset, where each file represents a different ISA. The query file is then classified as belonging to the ISA with the lowest Kullback–Leibler divergence.

The *CpuRec* [3] dataset is valuable for this research because it contains executable files from 77 different ISAs, each represented by a single code-only binary program. Although smaller than the IsaDetect dataset from [4], it has broad coverage of diverse ISAs.

Our use of endianness features builds on the work of Clemens [2], who found that a histogram of bigrams yields information about endianness due to the contained information on byte-adjacency. However, the space of all possible bigrams yields a very large feature space ($256^2 = 65536$), which introduces a problem known as the *curse of dimensionality*. The authors handle this problem by using a limited set of four bigrams: `0xfffe`, `0xfeff`, `0x0001` and `0x0100`.

3 Methodology

This paper proposes machine learning to classify specific characteristics of ISAs in binary programs. We achieve this by engineering features that extract relevant information from the binary code. These features are then used to train and test machine learning models, enabling the identification of key ISA characteristics. To ensure unbiased testing, we use Leave-One-Group-Out Cross Validation (LOGOCV). LOGOCV involves training a separate model for each ISA in the dataset, where the specific ISA is left out of the training data and used only for testing. This approach allows us to simulate the real-world scenario of encountering a previously unseen ISA.

This paper uses both the IsaDetectFull and CpuRec datasets. The IsaDetectFull dataset is used for endianness detection due to its large number of binary files and balance across big and little endianness. This dataset is favorable over the IsaDetectCode dataset as data affected by endianness also exists in parts of the binary file other than the code section (e.g., header information). The datasets and ISA characteristics are listed in Table 1.

CpuRec is used to detect fixed/variable instruction size and fixed instruction size, as it is more balanced across the classes these ISA characteristics contain than the IsaDetect datasets. However, CpuRec is still unbalanced for the fixed instruction size feature. Fixed instruction sizes tend to be 128 bits or less, and common sizes can be but are not limited to 8-, 16-, 24-, 32-, or 64-bits. This range of potential sizes makes it difficult to create balanced datasets, as some fixed instruction sizes will be less common than others.

Table 1: Architectures from the IsaDetectFull and CpuRec datasets with endianness (E) and instruction size (IS) characteristics (in bits). Blank table elements mean that particular characteristic was not available for the dataset and thus was not used for model training.

IsaDetectFull	**CpuRec**	**E**	**IS**	**IsaDetectFull**	**CpuRec**	**E**	**IS**
	6502	LE	8-32		MMIX	BE	32
	68HC08	BE	8-16		MN10300	LE	
	68HC11	BE	8-40		MSP430	LE	
	8051	LE	8-128		Mico32	BE	32
arm64	ARM64	LE	32		MicroBlaze	BI	32
	ARMeb	BE	32		Moxie	BI	32-48
armel	ARMel	LE	32		NDS32	BI	16-32
armhf	ARMhf	LE	32		NIOS-II	LE	32
	ARcompact	LE	16-32		PDP-11	LE	16
	AVR	LE	16-32		PIC10	LE	
alpha	Alpha	LE	32		PIC16	LE	
	AxisCris	LE	16		PIC18	LE	
	Blackfin	LE	16-32		PIC24	LE	24
	CLIPPER	LE	2-8	ppc64	PPCeb	BE	
	CUDA	LE	32	ppc64el	PPCel	LE	
	Cell-SPU	BE	32	riscv64	RISC-V	LE	32
	CompactRISC	LE	16		RL78	LE	
	Epiphany	LE	16-32		ROMP	BE	8-32
	FR30	BE	16		RX	LE	
	H8-300	BE	8-16	s390x		BE	
	H8S	BE		s390	S-390	BE	
hppa	HP-PA	BE	32	sparc	SPARC	BE	32
ia64	IA-64	LE	128	sparc64		BE	32
	IQ2000	BE			Stormy16	LE	
	M32R	BI	16-32		WASM	LE	
m68k	M68K	BE		amd64	X86-64	LE	8-120
	M88K	BI	32	i386	X86	LE	8-120
	MCore	BE	16		Xtensa	BI	16-24
mips64el		LE	32		Z80	LE	8-32
	MIPS16	BI	16	x32		LE	
mips	MIPSeb	BE	32	powerpc		BE	32
mipsel	MIPSel	LE	32	powerpcspe		BE	32

Experiments are conducted with the machine learning library `scikit-learn`. The selected classifiers are chosen to match those based on the paper from Kairajärvi et al. [4]. Using multiple classifiers allows for a more comprehensive evaluation of the data, as different models may capture different aspects of the underlying patterns.

Choosing hyperparameters for classifiers and parameters for engineered features is essential to producing well-performing models. These values are found using a grid search for the classifiers to which this applies (LogisticRegression and Support Vector Classifier). Following the approach from Kairajärvi et al. [4], various powers of ten are used to find parameters for classifiers. Powers of two are used to find the parameters for the engineered features, as the classes of the targeted ISA characteristics commonly are powers of two.

Model performance is evaluated by calculating the model accuracy, which is defined below in Equation 1.

$$model\_accuracy = \frac{correct\_classifications}{total\_classifications} \tag{1}$$

The model accuracy from Equation 1 above is used with every model created with LOGOCV to calculate the accuracy of a feature. Feature accuracy depends on the hyperparameters, specific classifier, and dataset. The equation for feature accuracy is shown below in Equation 2.

$$feature\_accuracy = \sum_{n=1}^{m} \frac{model\_accuracy_n}{m}, \tag{2}$$

where m is the number of ISAs (groups) used in the LOGOCV.

Feature accuracy is measured in comparison against a baseline. Generally, features performing better than the baseline contain data that is helpful in classifying the targeted ISA characteristic. These features perform better than a random guess, allowing them to assist in documenting the ISA. The baseline is defined as the ratio resulting from classifying all samples as the most frequent class. The formula for the baseline is shown below in Equation 3.

$$baseline = \frac{most\ frequent\ class\ count}{all\ count} \tag{3}$$

Detection of endianness focuses on differentiating binary files of big and little endianness. Two features are used for endianness detection, both inspired by previous research. The first of these is the Bigrams feature, which contains the frequency of every bigram from `0x0000` to `0xffff`. The second is the EndiannessSignatures feature, which includes the four selected bigrams from Clemens' [2] study (see Section 2): `0xfffe`, `0xfeff`, `0x0001` and `0x0100`. Due to the large dimensionality resulting from the full bigrams feature (256^2), only 100 binary files are used per ISA from the IsaDetectFull dataset when training and testing with this feature. This is to reduce the computational cost of training at the expense of the risk of overfitting data due to the curse of dimensionality. In practice this should not be a problem because the LOGOCV ensures an entire ISA is left out and used for testing. In contrast, all binary files are used with the EndiannessSignatures feature.

A feature using autocorrelation has been engineered to detect fixed/variable instruction size and fixed instruction size. This feature is named AutoCorrelation and is defined in Equation 4 to 6 below.

The computation of the AutoCorrelation feature is based on the Pearson correlation $r(x, y)$, defined in Equation 4:

$$r(x,y) = \frac{n\sum xy - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}, \tag{4}$$

where x and y are the variables (original and lag windows) and n refers to the number of samples in x and y. The autocorrelation function is computed using the `pandas.autocorr` function from the `pandas` library in Python. Its mathematical formulation is shown in Equation 5, where s represents the series of bytes in a given binary file, and k represents the given lag:

$$f(k) = r(\{s_i \mid 1 \leq i \leq (|s| - k)\}, \{s_j \mid k \leq j \leq |s|\}) \tag{5}$$

The autocorrelation values, $f(k)$, are used to calculate the AutoCorrelation feature, as shown in Equation 6, where l is the lag parameter of the AutoCorrelation feature, corresponding to the max lag used when calculating the autocorrelation values $f(k)$:

$$\text{AutoCorrelation} = \{f(k) \mid 1 \leq k \leq l\} \tag{6}$$

The AutoCorrelation feature shown above in Equation 6 calculates autocorrelation values by calculating the Pearson correlation $r(x, y)$ for a binary file and a specified range of lagged versions of itself. The aim is to discover periodicity resulting from repetitions of full or partial instructions at regular intervals corresponding to a multiple of the underlying fixed instruction size. In other words, we would expect autocorrelation peaks for lags that are an integer multiple of the fixed instruction size. For ISAs with fixed instruction sizes, this would result in a series with a general periodicity equal to the instruction size in bytes. The same would not be true for ISAs with variable instruction sizes, allowing for discrimination between these two classes.

Specifically, the AutoCorrelation feature is first used to determine whether the ISA has a fixed or variable instruction size. If the ISA is classified as having fixed-size instructions, the AutoCorrelation feature is then used to determine the specific instruction size by analyzing the periodicity in the autocorrelation values.

4 Results

4.1 Experimental Setup

All experiments were run on an AMD EPYC 7742 64-Core server with 128GB RAM running Debian Linux 11 (bullseye) and Linux kernel 5.10.0-22-amd64. Versions of the software libraries were Python 3.9.2, scikit-learn 1.3.2, SciPy 1.6.0, and pandas 1.1.5. The source code used in the experiments is publicly available on GitHub[3].

[3] https://github.com/joffe97/isa_detection (commit de355e0)

LogisticRegression and *Support Vector Classifier* (*SVC*) have a regularization parameter C which we tuned via grid search. The results of the hyperparameter tuning are presented in Table 2.

Table 2: Hyperparameter values of C for LogisticRegression (LR) and Support Vector Classifier (SVC) discovered via grid search.

ISA characteristic	**Data feature**	**LR** C	**SVC** C
Endianness	EndiannessSignatures	10^10	10^11
	Bigrams	10^5	10^3
Fixed/variable instruction size	AutoCorrelation	10^0	10^0
Fixed instruction size	AutoCorrelation	10^1	10^1

The tuned lag parameters of the AutoCorrelation feature for every configuration of classifier and ISA characteristic are shown below in Table 3. Each row in the table consists of lags tuned for configurations of classifiers and ISA characteristics, where the first column is the classifier, and the second and third columns show the tuned lags of the ISA characteristic.

Table 3: Tuned lag parameters for the AutoCorrelation feature.

Classifier	**Lag for fixed/variable instruction size**	**Lag for fixed instruction size**
1NeighborsClassifier	256	32
3NeighborsClassifier	256	128
5NeighborsClassifier	512	512
DecisionTreeClassifier	128	128
GaussianNB	32	256
LogisticRegression	128	128
MLPClassifier	1024	32
RandomForestClassifier	256	256
SVC	128	64

The baselines for the three targeted ISA characteristics are shown below in Table 4. Note that the baselines for the EndiannessSignatures and Bigrams features targeting endianness differ; Bigrams uses the same number of files per ISA (100), in contrast to EndiannessSignatures (full dataset). The first two columns contain ISA characteristics and classifier data features, respectively, where each row represents a configuration used in the experiments, while the third column contains the baseline result accuracy for each configuration.

Table 4: Feature baselines.

ISA characteristic	Data feature	Baseline accuracy
Endianness	EndiannessSignatures	0.556
	Bigrams	0.545
Fixed/variable instruction size	AutoCorrelation	0.581
Fixed instruction size	AutoCorrelation	0.680

4.2 Experiments

Endianness results are presented below in Figure 1. As mentioned in Section 3, these models are trained on the IsaDetectFull dataset, where EndiannessSignatures uses all files, and Bigrams uses 100 per ISA.

The results show that the EndiannessSignatures feature generally performs better than the Bigrams feature when detecting endianness. All models perform better than the baselines presented in Table 4 of 0.556 for EndiannessSignatures and 0.545 for Bigrams. The best-performing models for EndiannessSignatures and Bigrams achieve an accuracy of 0.994 and 0.986, respectively.

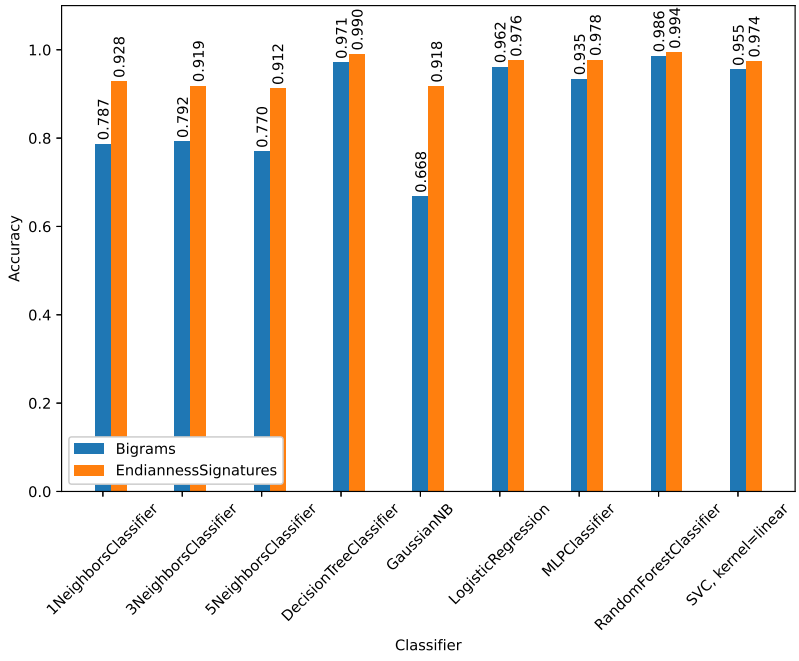

Fig. 1: EndiannessSignatures and Bigrams models targeting endianness (IsaDetectFull dataset).

Fixed/variable instruction size results are presented below in Figure 2. The plot in this figure is generated by calculating the average accuracy of the ISAs from the CpuRec dataset, grouped by fixed and variable instruction sizes. Lags in the x-axis represent numbers of bytes.

The plot shows that the AutoCorrelation feature can differentiate between ISAs of fixed and variable instruction sizes, as the plotted values show a clear difference between the two classes. ISAs of fixed instruction sizes are shown to have greater peaks than ISAs of variable instruction sizes at every fourth byte lag. This observed difference indicates that the AutoCorrelation feature is suitable for differentiation between fixed and variable instruction sizes.

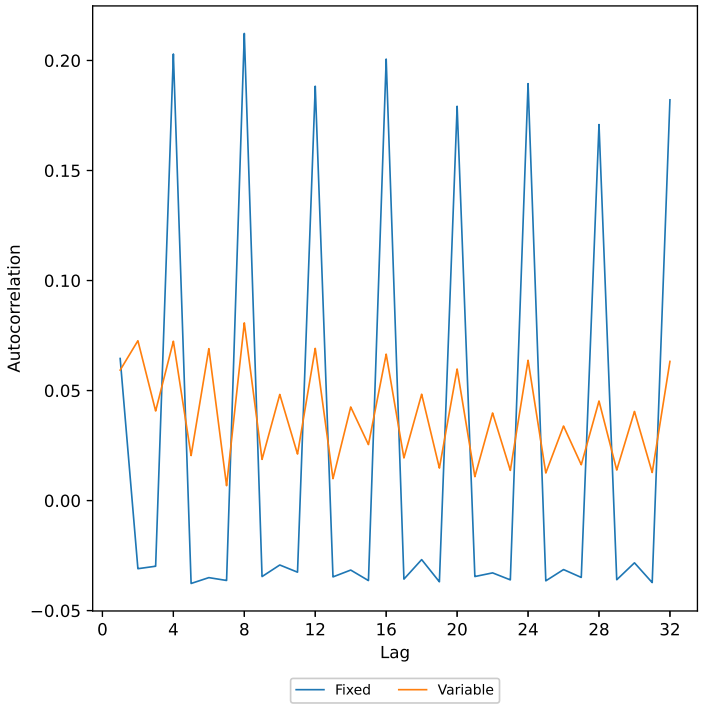

Fig. 2: AutoCorrelation mean values for fixed and variable instruction sizes (CpuRec dataset).

Figure 3 below shows that the AutoCorrelation feature can classify fixed/variable instruction size with an accuracy of 0.860, which is significantly greater than the baseline of 0.581.

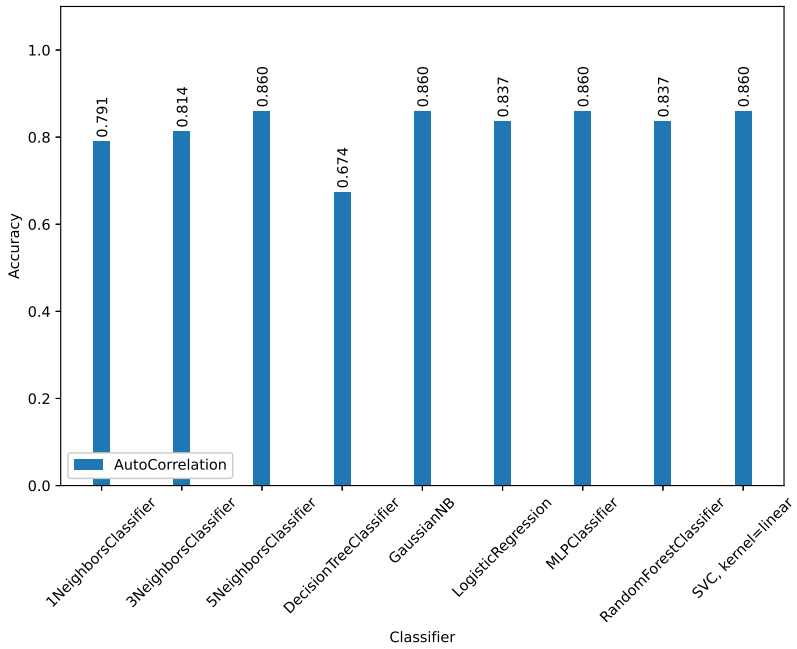

Fig. 3: AutoCorrelation models targeting fixed/variable instruction size (CpuRec dataset).

Fixed instruction size results are presented below in Figure 4. The values in this plot are generated similarly to the plot for fixed/variable instruction size in Figure 2. The average values calculated from the AutoCorrelation feature are grouped and plotted for a range of lags.

The plot shows there are generally peaks on lags equal to an integer multiple of classes' instruction sizes in bytes. This implies that the AutoCorrelation feature might also be suitable for fixed instruction size detection. The exception to this pattern is the data for 24-bit instruction size. It should be noted that only one file from a single ISA in the CpuRec dataset has an instruction size of 24 bits. This means the irregularity could result from a unique property of the specific binary file or ISA unrelated to instruction size.

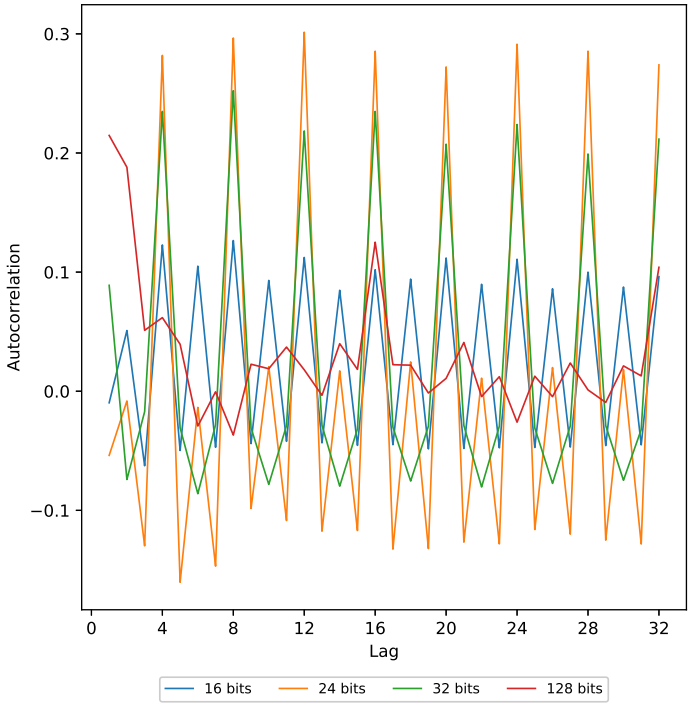

Fig. 4: AutoCorrelation mean values for fixed instruction sizes (CpuRec dataset).

Accuracies of the classifiers using the AutoCorrelation features in fixed instruction size detection are shown below in Figure 5. This Figure shows that the models can detect fixed instruction size with an accuracy of 0.880. This is significantly greater than the baseline of 0.680, meaning that the AutoCorrelation feature is suitable for classifying fixed instruction size.

It should be noted that the CpuRec dataset contains only one ISA each of 24 and 128 bits. Due to the use of LOGOCV, fixed instruction sizes belonging to only one ISA will not be able to be detected when generating results. Specifically, models testing one of these ISAs will not be trained on any instruction size of the same ISA, leading to the models being unable to classify it. This means that 2 of 77 ISAs will always be classified incorrectly. This should be considered in the analysis of resulting accuracies when targeting fixed instruction size.

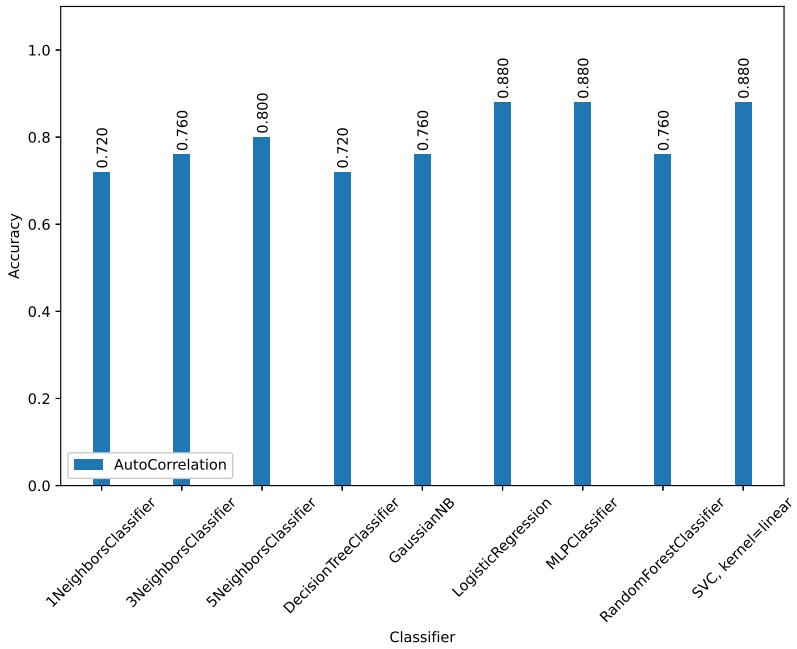

Fig. 5: AutoCorrelation models targeting fixed instruction size (CpuRec dataset).

5 Discussion

Regarding RQ1, the results from Section 4 show that machine learning can reliably detect ISA characteristics from binary programs by using features containing information related to the targeted ISA characteristics. The results show this to be true for the ISA characteristics targeted in this research, showing that models consistently achieve accuracies well above the baseline. The experiments show that the classifiers' accuracies vary based on the targeted ISA characteristic. For example, classifiers that perform well in detecting endianness, such as the Decision Tree Classifier, do not necessarily achieve high accuracy in detecting fixed/variable instruction size or fixed instruction size. This difference in performance highlights the benefit of running experiments with multiple classifiers.

Answering RQ2 requires analyzing the results presented in Section 4 to determine whether the ISA characteristics are detected reliably across a wide range of ISAs. The results show that we are able to detect endianness, fixed/variable instruction size, and fixed instruction size consistently with an accuracy greater than the baseline. Since the experiments use datasets with broad ranges of ISAs, we can conclude that the proposed approaches do lead to reliable detection of ISA characteristics across a wide range of ISAs.

The results from endianness detection show that bigrams can detect endianness from binary programs with any arbitrary ISA, with a greater accuracy than the baseline.

The experiments also demonstrate that the four bigrams included in the EndiannessSignatures feature further improve the accuracy, inferring that excluding non-descriptive bigrams increases accuracy. Although the curse of dimensionality cannot be ruled out as a factor, LOGOCV ensures that any overfitting means the results are conservative (tending towards poorer accuracy than better because the target ISA has not been seen during training).

The experiments show that features using autocorrelation can differentiate ISAs of fixed and variable instruction sizes and classify specific fixed instruction sizes with greater accuracy than the baseline. This demonstrates that signal processing is applicable in the field of reverse engineering for ISA classification.

If we presume that correctly classifying a binary program into one of a set of known ISAs also yields the endianness or instruction size, then our results are comparable to those of Kairajärvi et al. [4]. Kairajärvi et al. can correctly assign a known ISA to a binary program with an accuracy of 98%. The results from this paper show that the proposed methods are not able to retrieve ISA characteristics with the same accuracy, as the chance of false classifications accumulates for every ISA characteristic detection. This shows that the methods proposed by Kairajärvi et al. are more suited for detecting ISA characteristics when the ISA is known. As mentioned, the method of Kairajärvi et al. cannot detect ISA characteristics for unknown ISAs. The results presented in Section 4 show that the methods utilized in this paper can do so with great accuracy. These methods are, therefore, more suitable for detecting ISA characteristics where the ISA is unknown.

6 Conclusion

This work presents various features and methods for detecting endianness and instruction size characteristics in binary programs with unknown ISAs. Experiments with features using bigrams are performed, demonstrating that they are well suited for endianness detection of binary programs with unknown ISAs. Fixed/variable instruction size and fixed instruction size are also shown to be accurately predicted for such binary programs using a feature that use autocorrelation.

Conducting experiments using a range of classifiers from the `scikit-learn` library in Python has proven valuable, as results demonstrate that classifier choice significantly impacts accuracy, enabling the application of models best suited for specific classifications.

The use of Leave-One-Group-Out Cross-Validation trains a model without any direct knowledge of the targeted binary program's ISA. This ensures experiments that simulate the detection of ISA characteristics for truly unknown ISAs.

Several possibilities exist for future work. One involves creating a more extensive balanced dataset, which would lead to more robust experiments. More training data also allows models to discover more detailed patterns related to the various target classes. Balancing this dataset leads to models being more equally influenced by all classes instead of favoring their ability to detect the most frequently occurring class.

The disadvantage of unbalanced datasets could be mitigated for code-only binary files by splitting them into smaller chunks and training the models on an equal number of chunks from each target class. Future work could explore whether this approach improves accuracy for under-represented classes. However, care should be taken to avoid splitting instructions at non-boundary locations, which could negatively impact results.

Extending the research by developing additional methods for discovering other ISA characteristics would further streamline the reverse engineering of binary files with unknown or undocumented ISAs. Suggested ISA characteristics include word size, register count, instruction format, opcode encoding, and subroutine boundaries (e.g., isolation of CALL/RET opcodes as in [7]).

References

1. Arm: Glossary - instruction set architecture (isa) (2024), https://www.arm.com/glossary/isa
2. Clemens, J.: Automatic classification of object code using machine learning (2015)
3. Granboulan, L.: cpu_rec (6 2024), https://github.com/airbus-seclab/cpu_rec
4. Kairajärvi, S., Costin, A., Hämäläinen, T.: Isadetect: Usable automated detection of cpu architecture and endianness for executable binary files and object code (2020), https://doi.org/10.1145/3374664.3375742
5. Kairajärvi, S., Costin, A., Hämäläinen, T.: Towards usable automated detection of cpu architecture and endianness for arbitrary binary files and object code sequences (2019)
6. Nicolao, P.D., Pogliani, M., Polino, M., Carminati, M., Quarta, D., Zanero, S.: Elisa: Eliciting isa of raw binaries for fine-grained code and data separation (2018)
7. Pettersen, H., Morrison, D.: Call graph discovery in binary programs from unknown instruction set architectures (2024), https://arxiv.org/abs/2401.07565
8. Sahabandu, D., Mertoguno, S., Poovendran, R.: A natural language processing approach for instruction set architecture identification (2022)
9. TIS Committee: Executable and linking format (elf) (1995), https://refspecs.linuxfoundation.org/elf/elf.pdf