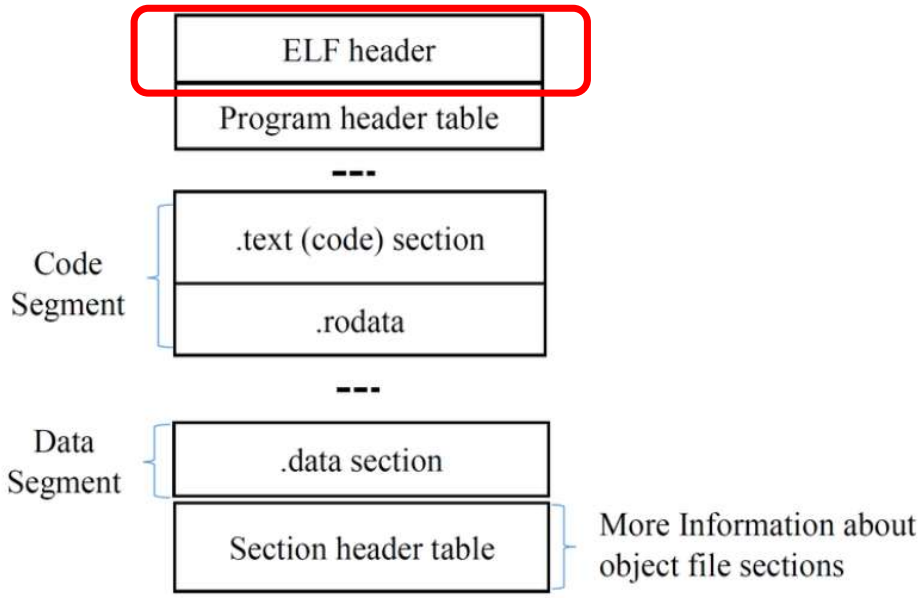
# ProvSpider: A Robust Toolkit for Binary Provenance Analysis

- Presenter: Zhiwei Fu

McGill

# Binary

- Normally, descriptive information (“origin”) about one binary, including **CPU architecture** (x86, ARM, MIPS, etc.), **bitness** (32-bit or 64-bit), **endianness** (big or little), etc., can be found directly within the file headers.
- **The very first step** of many reverse engineering-based methods, such as vulnerability detection, clone search, and malware detection.




McGill

# *Origin* is not always available...

(1) embedded software use non-standard / custom file format

(2) Resource-constrained IoT devices strip metadata for saving space and simplify execution.

→ Disassemblers require analysts repeatedly manual attempt to *guess* the correct CPU architecture.

We need **Binary Provenance Analysis** to *recover* the “origin” of a binary purely from its **raw bytes.**

McGill

# Challenges

- C1: Diverse and Legacy CPU Architectures

  Different architectures have **unique** instruction encodings, addressing modes, etc.. **Legacy** architectures (like S200 and HP-PA) are hard to collect, making training data imbalanced.

- C2: Extremely Long Byte Sequences

  Raw binaries can span **millions** of bytes.

- C3: Code Compilation

  Compilation is a **lossy** process in which only basic low-level instructions and data representations understood by the target CPU are preserved.

McGill

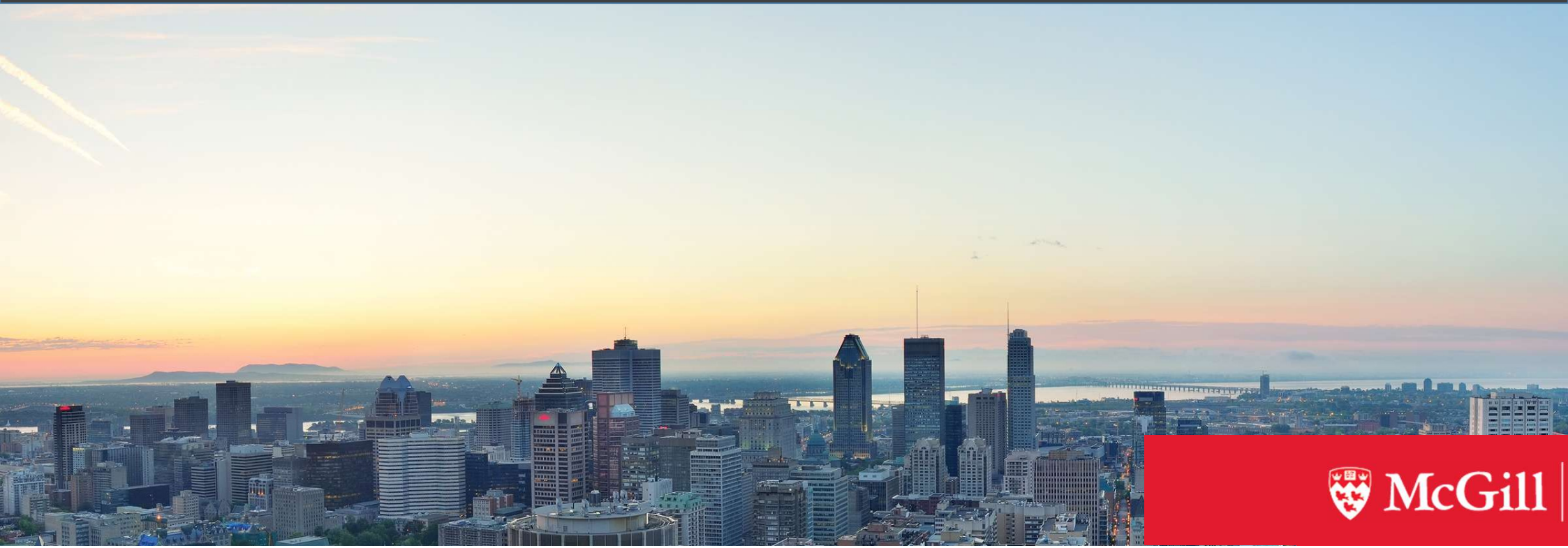
# Related Work

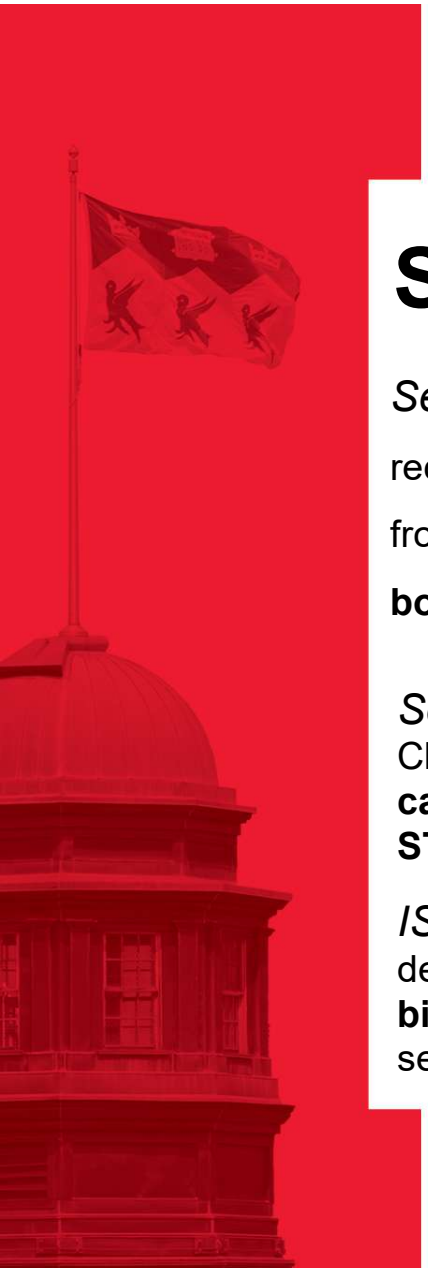
- CPU_Rec [1]
  - A **rule-based pattern-matching** method for CPU architecture recognition. It extracts **architectural signature features** and match by comparing Kullback-Leibler divergence.
  - *Weaknesses*: Heavily relies on manual extraction; performance is limited to data quality; only perform CPU recoginition
- ISAdetect [2]
  - A **random-forest classifier** trained on **293 handcrafted features** to identify CPU architecture, bitness, and endianness (Instruction Set Architecture).
  - *Weakness*: Relies on manual feature extraction; preforms inconsistently across architectures.

McGill

# ProvSpider

- Objective: develop a robust toolkit that (1) detects segment boundaries, (2) classifies segment types, and (3) identifies ISA directly from raw bytes against three challenges.





# Sequential Analysis

*Segment boundary detection*

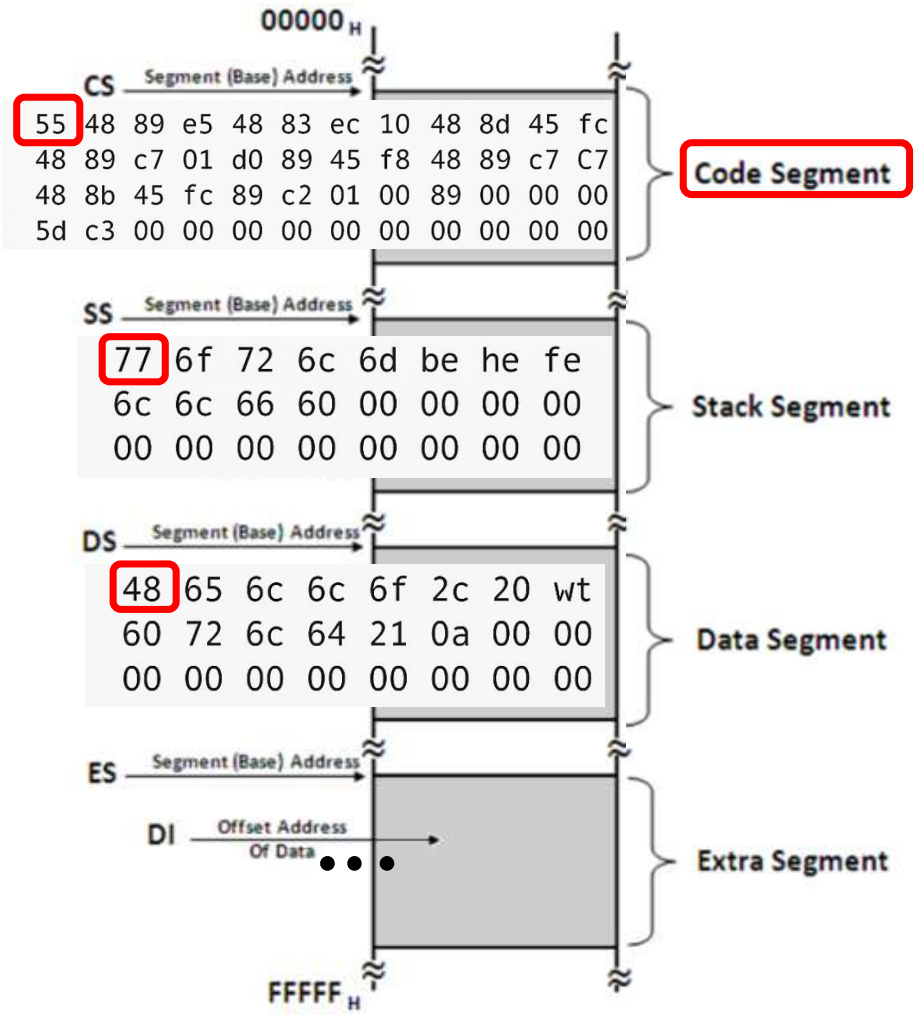
reconstruct the structure of executables directly from byte sequences; Each byte is labeled as **boundary (1)** or **non-boundary (0)**

*Segment type detection*

Classifies every detected segment into one of **categories** (**CODE, DATA, CONST, BSS, STACK, etc.**)

*ISA identification*

determines the binary's **CPU architecture, bitness, and endianness** based on the **CODE** segment




McGill

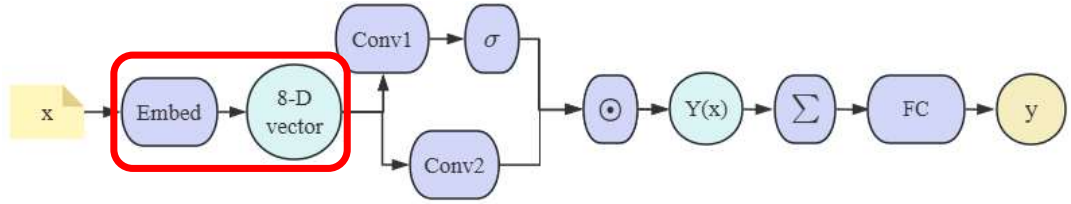
# Sliding Window – C2

- Partition byte sequences into fixed-length chunks.
- Zero-pad shorter chunks
- Filter out noisy chunks contains only {0, 255}
- Larger windows reduce batch parallelism and increase padding overhead for shorter chunks, whereas smaller windows make it difficult to capture global dependencies → trade-off: *"chunk_size"* = 5,000

**Algorithm 1** Sliding Window for an Executable

```
Initialize chunk_size ← 5000
Initialize dataset_c ← ∅
for start ← 0 to len(E) step chunk_size do
    end ← start + chunk_size
    chunk ← E[start : end]
    if len(chunk) < chunk_size then
        crunk ← zero-pad(chunk, chunk_size)
    end if
    if unique(chunk) ⊈ {0, 255} then
        Append chunk to dataset_c
    end if
end for
```

McGill



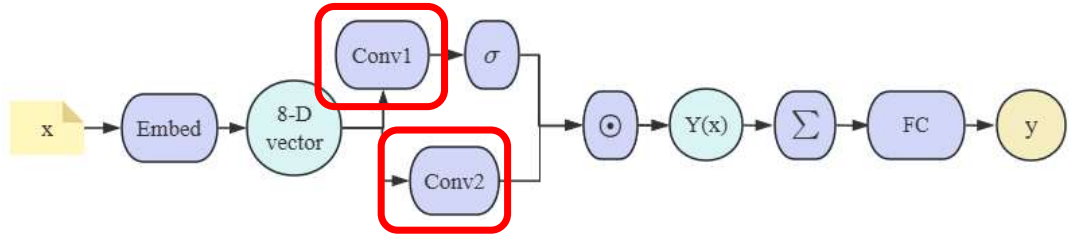

# *Spider* Models – C1 & C3

- Three spider models (CNN based) for each task, sharing a general architecture. Goal is to learn a robust *f* mapping the input raw bytes (*x*) to labels (*y*) and *K* varies based on tasks.

$$f : x \longrightarrow y \in \{1, \ldots, K\}.$$

- Convert raw bytes to *grey-scale image* (0~255).
- However, *0x10* isn't "closer" to *0x11* than to *0xA0*, so we embed bytes' values into 8-dimensional vectors instead of treating them as integers. Semantically similar bytes' embeddings / vectors will become **closer** in feature space during backpropagation, enabling models to capture **long-range / cross-byte dependencies** between different code segments.

McGill



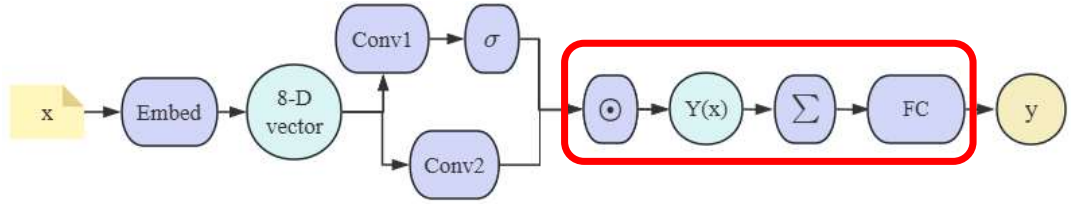

# *Spider* Models – C1 & C3

- Three spider models (CNN based) for each task, sharing a general architecture. Goal is to learn a robust *f* mapping the input raw bytes (*x*) to labels (*y*) and *K* varies based on tasks.

$$f : x \longrightarrow y \in \{1, \ldots, K\}.$$

- Two *1D-Conv* filters slide across byte sequences to capture **short-range pattern** as it preserves the spatial order of bytes and can process long sequences in parallel (immune to vanishing gradients like RNNs).

McGill



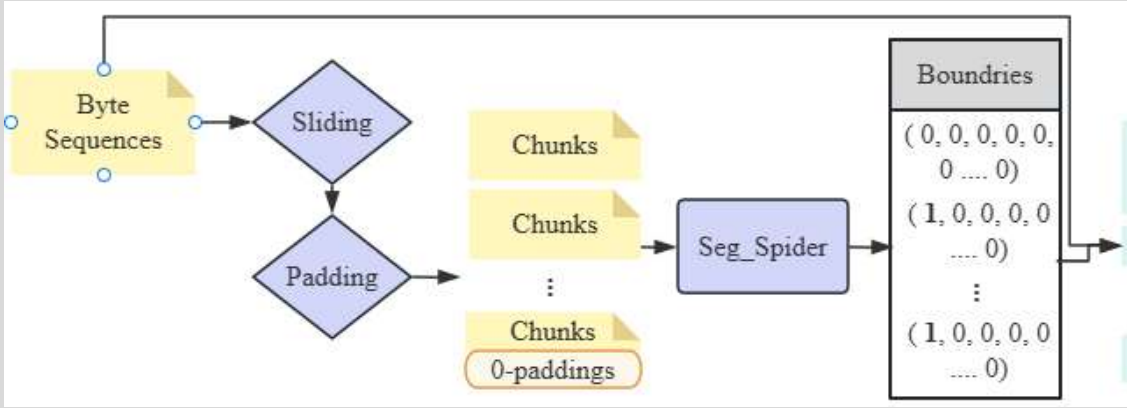

# Gate Mechanism

- Gate Mechanism: After convolution, two feature maps are generated; *Conv1* passes through a **sigmoid** activation to create “gating map” (values between 0 and 1). Gates are **multiplied element-wise** with the *Conv2*’s feature map to strengthen important signals and suppress irrelevant bytes or padding.
- The gate output *Y(x)* is a matrix of shape [Channels × Length] Each channel corresponds to one convolutional filter capturing a different local pattern. To capture the **overall activation strength** across the entire input sequence, we sum activations across the length dimension.
- A fully connected layer project the aggregated vectors to logits.

McGill

# Train & Test

- Training:  Each Spider model follows a mini-batch g
  epochs. Parameters are updated by backpropagatic
  epochs, the model undergoes a validation phase.
- Loss function: For binary classification tasks, such a
  use BinaryCrossEntropy; for multi-label classificatio

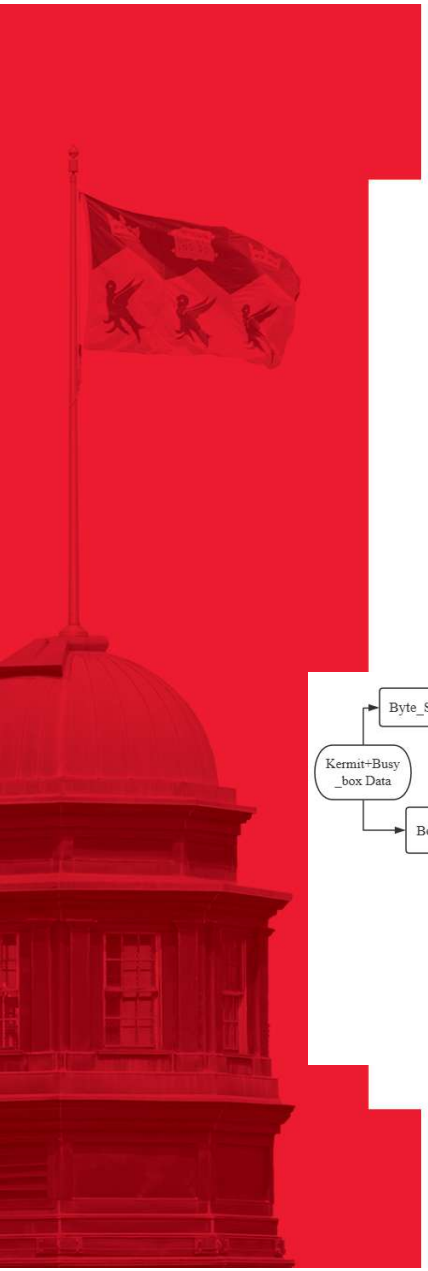



** busybox_1.16.1_busybox-armv5l

| Address | Class | Achitecture | Endianness | Bits |
|---|---|---|---|---|
| [0, 180) | CODE | LOAD | b32 | le |
| [180, 208) | CODE | LOAD | b32 | le |
| [208, 936572) | CODE | ARM | b32 | le |
| [936572, 936592) | CODE | LOAD | b32 | le |
| [936592, 1115156) | CONST | N/A | N/A | N/A |
| [1115156, 1115180) | DATA | N/A | N/A | N/A |
| [1115180, 1118208) | CONST | N/A | N/A | N/A |
| [1118208, 1118212) | DATA | N/A | N/A | N/A |
| [1118212, 1118216) | DATA | N/A | N/A | N/A |
| [1118216, 1118220) | DATA | N/A | N/A | N/A |
| [1118220, 1118224) | DATA | N/A | N/A | N/A |
| [1118224, 1118544) | DATA | N/A | N/A | N/A |
| [1118544, 1118792) | DATA | N/A | N/A | N/A |
| [1118792, 1120536) | CODE | LOAD | b64 | le |

** cku196.linux-m68k-db2.2

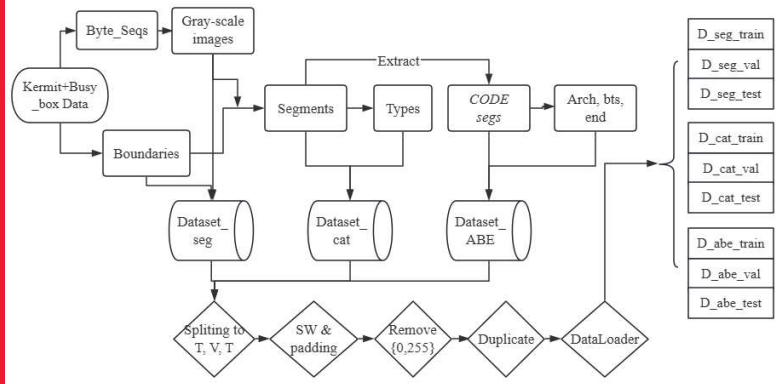
| Address | Class | Achitecture | Endianness | Bits |
|---|---|---|---|---|
| [0, 9172) | CODE | LOAD | b32 | le |
| [9172, 9198) | DATA | N/A | N/A | N/A |
| [9198, 9200) | DATA | N/A | N/A | N/A |
| [9200, 12720) | CODE | LOAD | b32 | be |
| [12720, 747056) | CODE | 68K | b32 | be |
| [747056, 747070) | DATA | N/A | N/A | N/A |
| [747070, 1115784) | CONST | N/A | N/A | N/A |
| [1115784, 1326192) | DATA | N/A | N/A | N/A |
| [1326192, 1326196) | DATA | N/A | N/A | N/A |
| [1326196, 1326204) | CODE | LOAD | b64 | le |
| [1326204, 1326212) | CODE | LOAD | b64 | le |
| [1326212, 1424023) | CODE | LOAD | b32 | be |



| Archs / Metrics |
|---|
| 68K |
| Alpha |
| ARM |
| HP-PA |
| IA64 |
| Metapc |
| MIPS |
| PDP-11 |
| S200 |
| SuperH |
| SPARC |
| VAX |

| Types / Metrics |
|---|
| CODE |
| DATA |
| CONST |
| BSS |
| STACK |
| UNK |
| TBL |
| ROOT |
| OVR |

# Experiment




Dataset

- Collect 3,691 real-world binaries from *BusyBox* and *Kermit* libraries, covering **12** different CPU architectures (IDA Pro).
- After segmentation, it yields in total of 15,253 segments of **9** segment types (GZIP).
- Generate train, validation, and test sets for each task using ratio of *70:10:20*.

McGill

# Benchmarking

- SOTA models:
  - Trained CPU_Rec [1] and ISAdetect [2] both publicly avaliable on GitHub.
- Evaluation:
  - **Accuracy** for classification tasks
  - **AUROC** (Area Under the Receiver Operating Characteristic Curve) calculates model's ability to distinguish classes, especially useful in multi-label classification.

McGill

# Performance

| Model | Segmentation | | Seg Classification | | CPU Recognition | | Bitness | | Endianness | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Accu | AUROC | Accu | AUROC | Accu | AUROC | Accu | AUROC | Accu | AUROC |
| CPU_REC [8] | N/A | N/A | N/A | N/A | 0.8306 | 0.9141 | N/A | N/A | N/A | N/A |
| ISADETECT [10] | N/A | N/A | N/A | N/A | 0.5220 | 0.7177 | 0.9580 | 0.9580 | 0.9655 | 0.9655 |
| ***ProvSpider*** | **0.9953** | **0.8652** | **0.7760** | **0.8176** | **0.9216** | **0.9948** | **0.9790** | **0.9925** | **0.9783** | **0.9961** |

ProvSpider achieves the **highest** accuracy and AUROC in all five tasks. In contrast, baselines CPU_rec [1] and ISAdetect [2] are designed only for CPU / ISA detection and perform worse than our model (CPU_rec = 83.06% acc; ISAdetect = 52.20% acc).

McGill

# Performance

| Model | Segmentation | | Seg Classification | | CPU Recognition | | Bitness | | Endianness | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Accu | AUROC | Accu | AUROC | Accu | AUROC | Accu | AUROC | Accu | AUROC |
| CPU_REC [8] | N/A | N/A | N/A | N/A | 0.8306 | 0.9141 | N/A | N/A | N/A | N/A |
| ISADETECT [10] | N/A | N/A | N/A | N/A | 0.5220 | 0.7177 | 0.9580 | 0.9580 | 0.9655 | 0.9655 |
| ***ProvSpider*** | **0.9953** | **0.8652** | **0.7760** | **0.8176** | **0.9216** | **0.9948** | **0.9790** | **0.9925** | **0.9783** | **0.9961** |

ProvSpider performs consistently across all architectures even deal with highly imbalanced dataset (only 10 VAX sample were collected).

However, rule-based and pattern-matching based models fail when detecting architectures that have less distinctive patterns.

| | *ProvSpider* | | CPU_REC [8] | | ISADETECT [10] | |
|---|---|---|---|---|---|---|
| Archs / Metrics | Accu | AUROC | Accu | AUROC | Accu | AUROC |
| 68K | 0.8031 | 0.9874 | 0.0000 | 0.5000 | 1.0000 | 0.9992 |
| Alpha | 0.9908 | 0.9999 | 1.0000 | 0.9886 | 1.0000 | 1.0000 |
| ARM | 0.9448 | 0.9953 | 1.0000 | 1.0000 | 0.0000 | 0.5000 |
| HP-PA | 0.9057 | 0.9938 | 1.0000 | 1.0000 | 0.0000 | 0.0000 |
| IA64 | 0.9940 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.9911 |
| Metapc | 0.9647 | 0.9891 | 0.9682 | 0.9848 | 0.9980 | 0.9899 |
| MIPS | 0.9675 | 0.9977 | 1.0000 | 1.0000 | 0.2667 | 0.6333 |
| PDP-11 | 0.9779 | 0.9999 | 1.0000 | 1.0000 | 0.0000 | 0.5000 |
| S200 | 0.9010 | 0.9950 | 0.0000 | 0.5000 | 0.0000 | 0.5000 |
| SuperH | 0.9254 | 0.9937 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| SPARC | 0.9268 | 0.9967 | 1.0000 | 1.0000 | 1.0000 | 0.9983 |
| VAX | 0.7641 | 0.9940 | 1.0000 | 0.9988 | 0.0000 | 0.5000 |

McGill

# Future Work

- Collect more diverse executable files and cover a broader range of CPU architectures.
- Explore more sophisticated attention mechanism to further boost performance.
- Design data-balancing strategies.

McGill

# Thanks

[1] L. Granboulan, S. Sterz, L. Brun, and V. Dary, “Airbus-seclab/cpu rec:Recognize cpu instructions in an arbitrary binary file,” Mar 2024.[Online]. Available: https://github.com/airbus-seclab/cpu rec

[2] S. Kairaj¨arvi, A. Costin, and T. H¨am¨al¨ainen, “Isadetect: Usable au-tomated detection of CPU architecture and endianness for executablebinary files and object code,” in CODASPY ’20: Tenth ACM Conferenceon Data and Application Security and Privacy, New Orleans, LA, USA,March 16-18, 2020, V. Roussev, B. Thuraisingham, B. Carminati, andM. Kantarcioglu, Eds. ACM, 2020, pp. 376–380.

McGill