IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Students:
Arthur Frey
Timothée Léveillé
Auguste Célérier

Supervisors:
Yehya Nasser
Mohammed Mezaouli

Project Falcon

Using machine learning models to detect malwares based on processors power traces and instruction traces

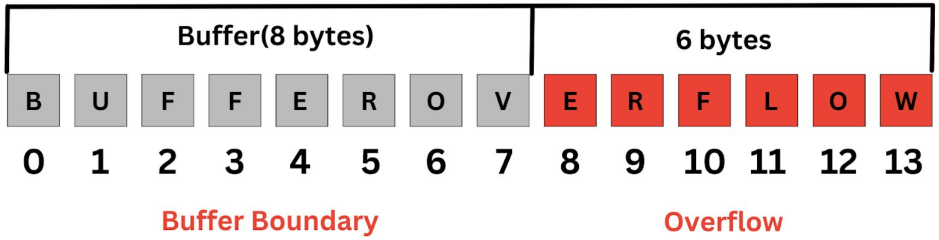
IoT devices are characterized by :

- **“ dedication to a single purpose or very few dedicated functions ”**
- **“ their use of nonstandard processors ”**
- **“ real-time operating systems to ensure that tasks meet timing and resource constraints ”**

Their are facing more and more threats : IoT malware attacks had jumped 400% from 2022 to 2023 (according a Zscaler report)

Problem : Signature detection with unique malware

DLL Hijacking Attack



Based on a processor consumption and instruction trace, detect if vulnerable functions are called

Ressources

We started our work by analysing three state-of-the-art papers:

Real-time instruction-level verification of remote IoT/CPS devices via side channels

Yunkai Bai · Jungmin Park · Mark Tehranipoor· Domenic Forte

Attentive transformer deep learning algorithm for intrusion detection on IoT systems using automatic Xplainable feature selection

Demóstenes Zegarra Rodríguez , Ogobuchi Daniel Okey, Siti Sarah Maidin, Ekikere Umoren Udo, João Henrique Kleinschmidt

Identification of Return-Oriented Programming Attacks Using RISC-V Instruction Trace Data
DANIEL F. KORANEK, SCOTT R. GRAHAM, BRETT J. BORGHETTI AND WAYNE C. HENRY

Data visualization

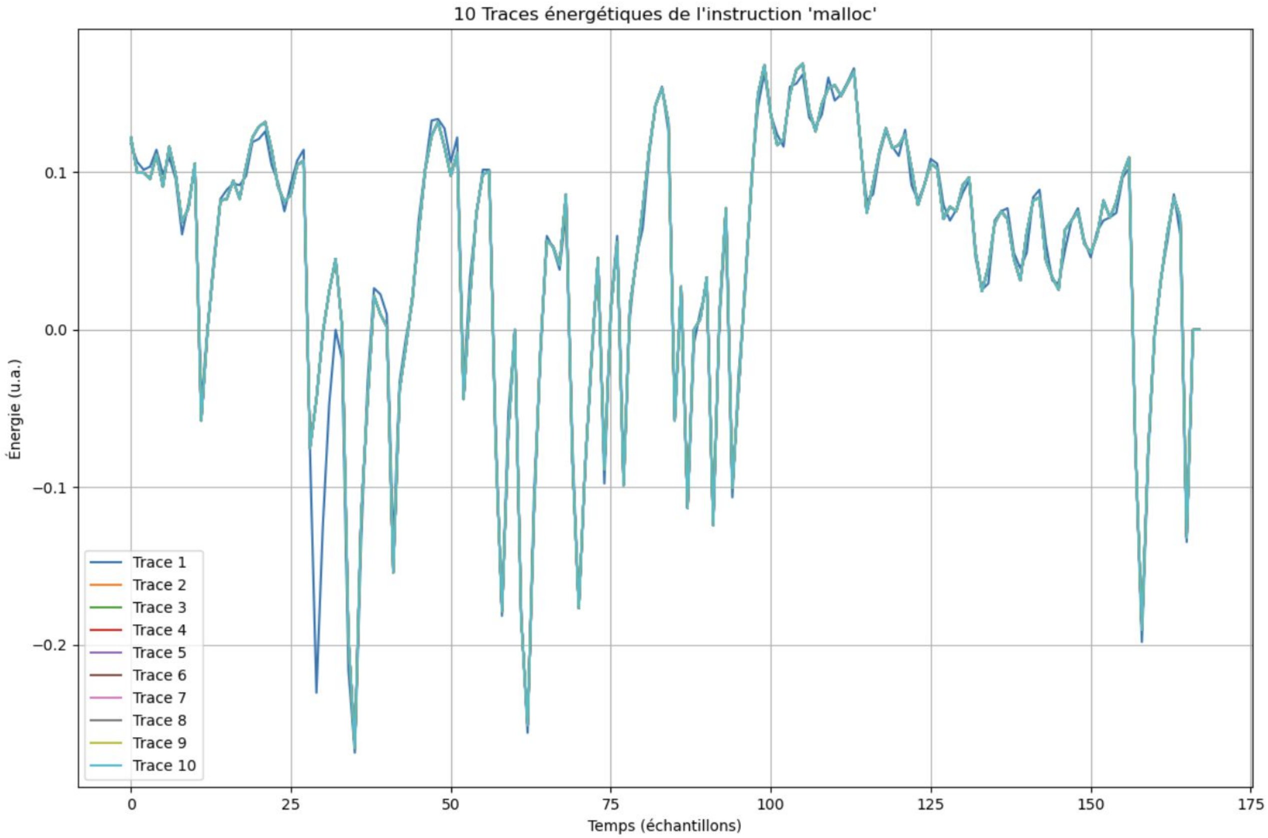
Power trace and instruction trace

Our dataset :

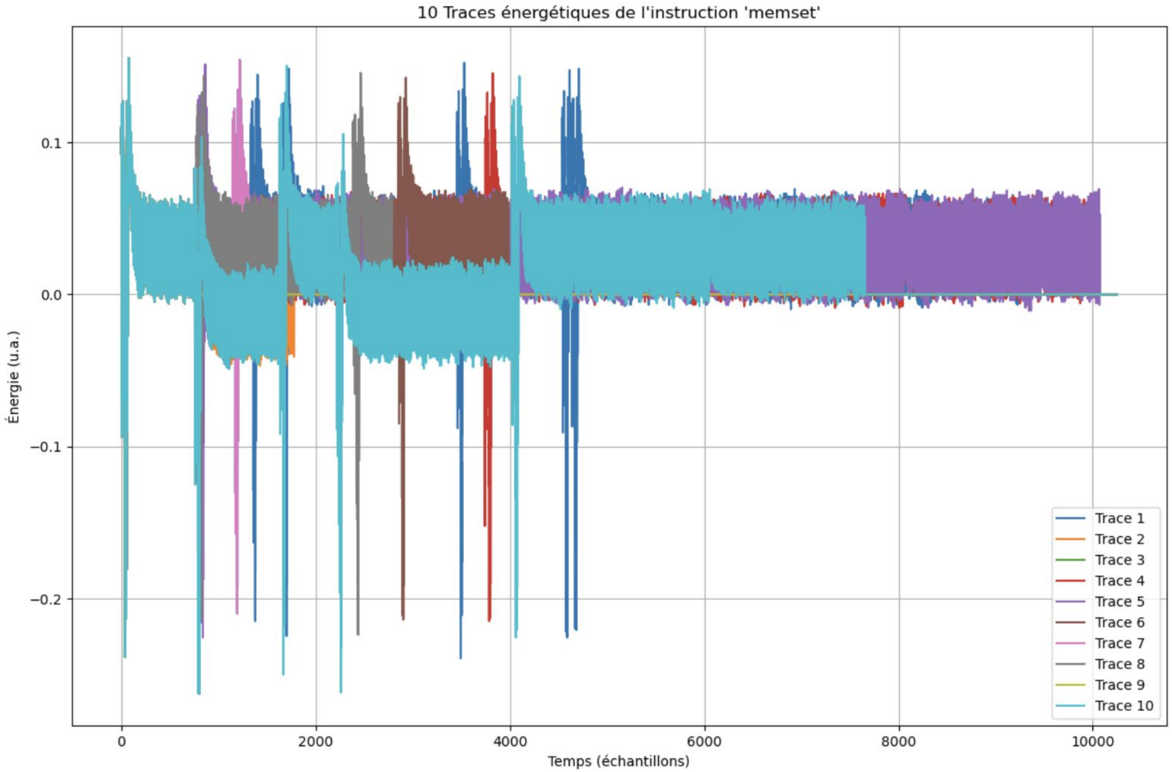
For power traces we had:

- 11 classes of functions
- around 3500 traces per class

Data Visualization



Data Visualization



How to detect such traces in an application power trace?

Main issues raised :

- For a given function, power traces can be longer or shorter

- It's hard to classify them with statistical methods or with classical machine learning methods (SVM, Random forests, etc..) cf *I know what you do* project.

What we need:

- A model that "looks" at patterns on the whole signal
- A model that understand the global context of those patterns

The transformer model

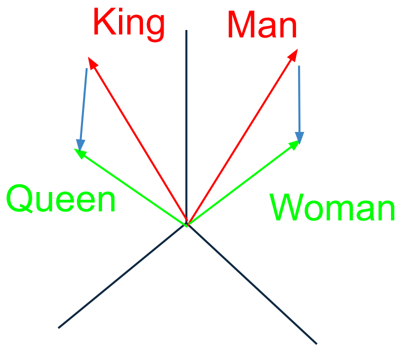
The transformer model originally designed for natural language allow us to:

- **capture global patterns**
- **handle variable length sequences**
- **highly parallelizable architecture**

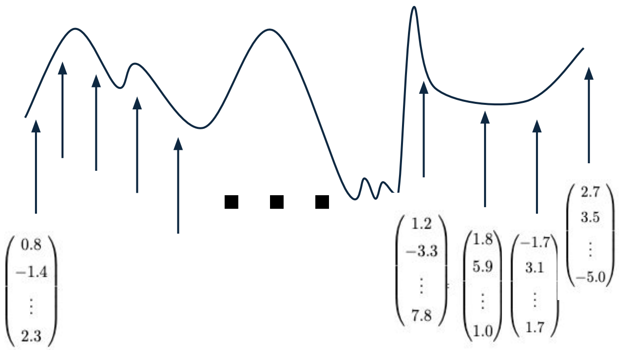
Values of the power trace are like words in a sentence

Each word of the sentence is encoded as high dimension vector
(12,288 dimension on GPT-3)

Directions have semantic meanings



We do the same thing for each value of the trace (dimension 128)



$$\begin{pmatrix} 2.7 \\ 3.5 \\ \vdots \\ -5.0 \end{pmatrix}$$

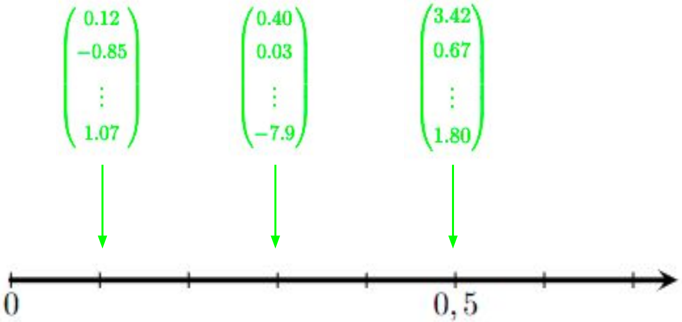
2.7: normalized power
3.5: similarity to a peak in calloc
.
.
.
-5.0: similarity to a decrease in free

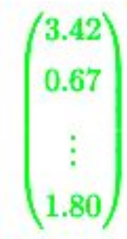
Transformer

Meaning is encoded

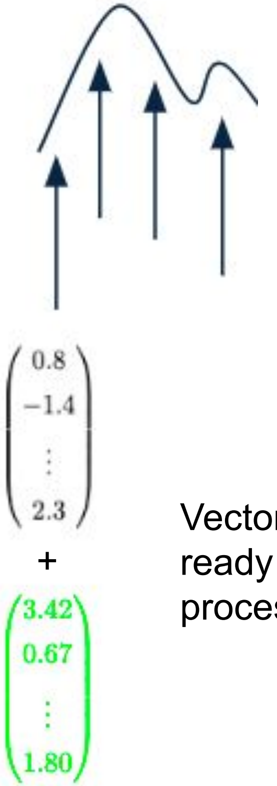
When processing a sentence/trace, we also need the position of the word/value to be encoded

We encode the positions with high-dimension vectors (128 also)





3.42: distance to the end
0.67: even-odd indicator
.
.
.
1.80: mid-point detector

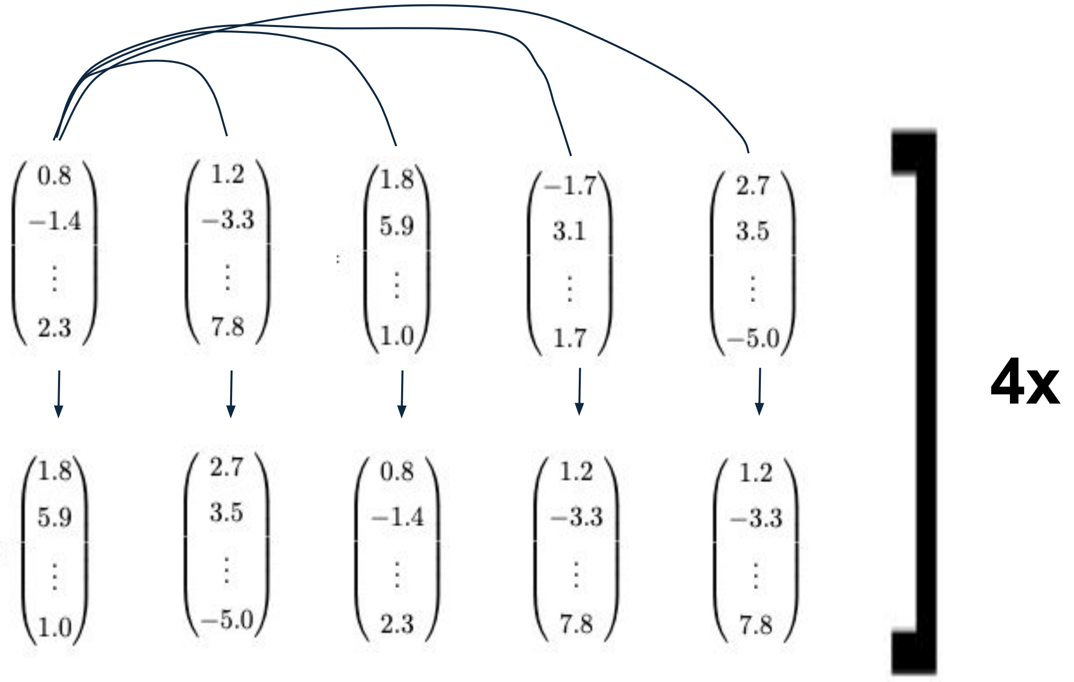


Vector ready to be processed

Vectors go through encoders blocks

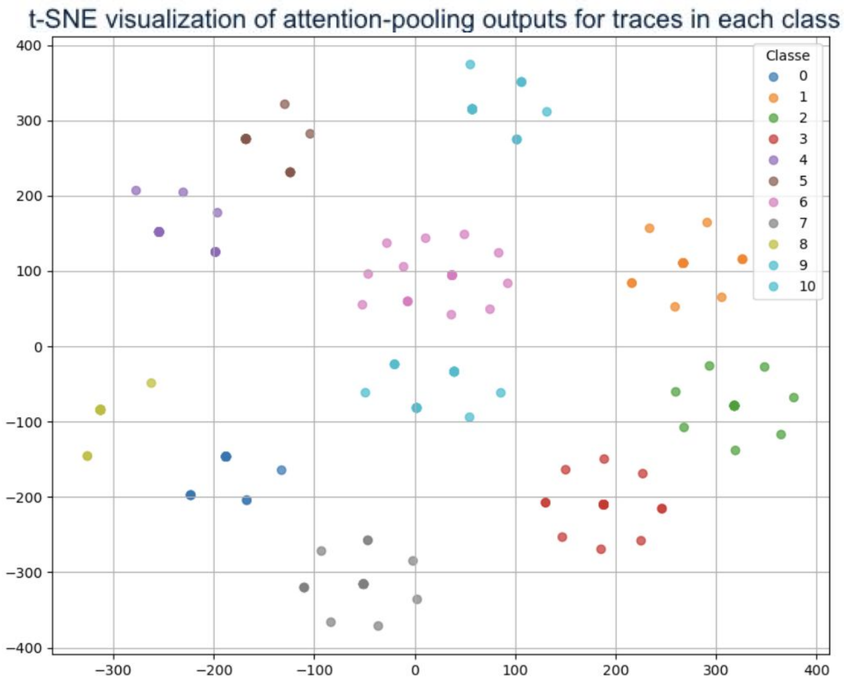
Self-Attention

Feed-Forward



Two last steps:

- Attention pooling: gives a unique vector for the trace capturing the most relevant information
- Classifier: associate logits to each class of function based on this unique vector



unique vector $\begin{pmatrix} 2.7 \\ 3.5 \\ \vdots \\ -5.0 \end{pmatrix}$ $\xrightarrow{\textit{classifier}}$ sprintf : 0.5, memset: -4, memcopy: 9, strncat: 3.4, … $\xrightarrow{\textit{softmax}}$ probability $\longrightarrow$ class prediction

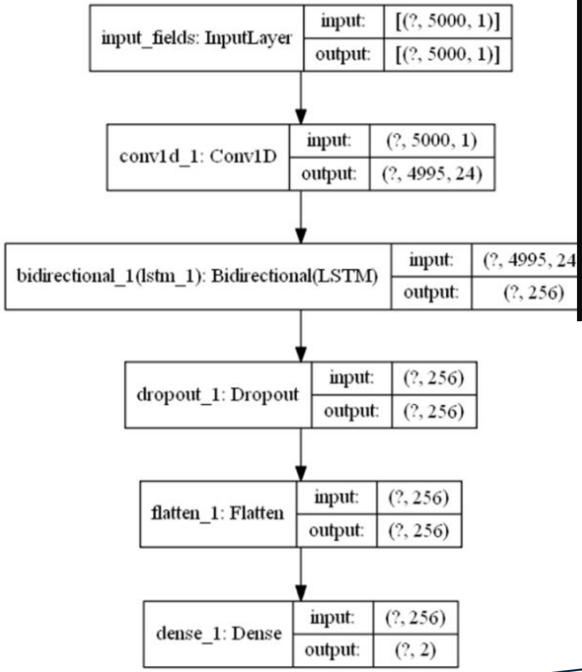
The LSTM model

From the data to the function

Based on Identification of Return-Oriented Programming Attacks Using RISC-V Instruction Trace Data

List of tokenized instructions

CSV Files

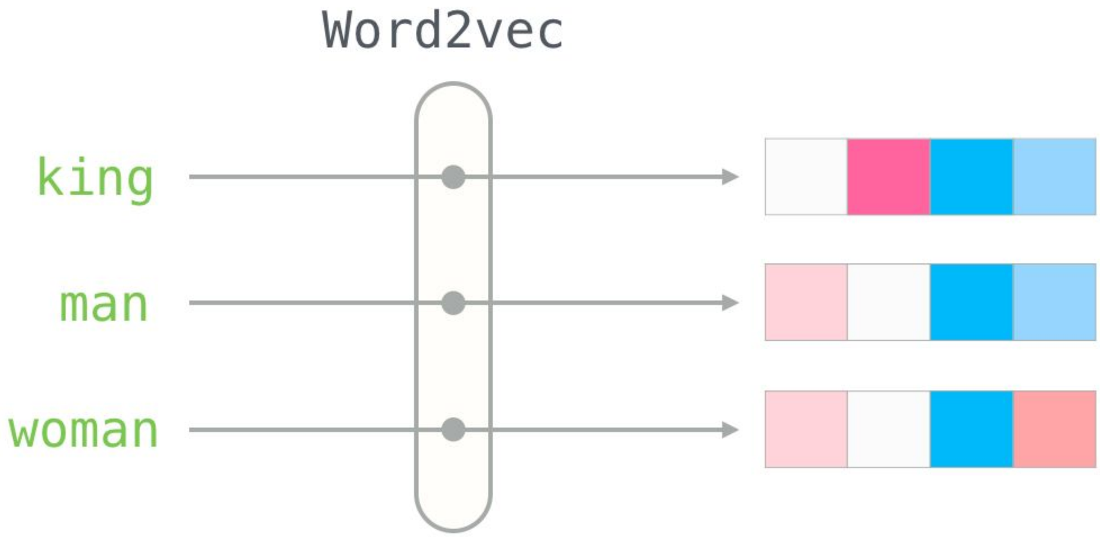



```
Sequences of combined length 1: [tensor([  2,   3,   4,   5,   5,   6,   7,   8,   4,   5,   5,   9,  10,  11,
         12,  13,  11,  14,  15,  16,  17,  18,  19,  20,  21,  11,  22,  23,
          7,  24,   4,   5,   5,  25,  26,  22,  27,  15,  22,  17,  28,  29,
         30,  31,  16,  16,  32,  33,  16,  34,  35,  36,  37,  38,  16,  14,
         39,  16,  40,  41,  42,  16,  17,  28,   4,  16,  43,  15,  16,  17,
         28,  44,  45,  21,  11,  46,  23,  47,  48,  26,  46,  49,   7,  50,
          4,   5,   5,  51,  26,  22,  52,  53,  54,  38,  22,  52,  55,  56,
         38,  16,  14,  57,  16,  22,  52,  58,  59,  31,  16,  16,  60,  15,
         61,  17,  35,  15,  22,  62,  35,  15,  16,  63,  44,  59,  21,  22,
         23,   7,  64,  13,  22,  14,  42,  16,  63,  42,  11,  62,  18,   4,
         16,  65,  57,  11,  11,  43,  15,  16,  63,  15,  11,  62,  18,   2,
          3,  66,  11,  67,   4,  68,  11,  27,  26,  16,  49,  69,  70,  38,
         11,  71,  72,  73,  38,  16,  49,  21,  16,  61,  74,  72,  75,  38,
         22,  52,  76,  77,  26,  61,  78,  26,  11,  65,  26,  79,  43,  15,
         22,  80,  36,  70,  81,  22,  82,  33,  22,  83,  35,  26,  84,  49,
         69,  85,  26,  86,  74,  26,  79,  87,  26,  88,  52,  38,  46,  87,
         15,  46,  83,  18,  89,  90,  91,  92,  26,  22,  78,  26,  61,  43,
         26,  11,  93,  15,  16,  62,  94,  44,  95,  21,  22,  23,   7,  64,
         13,  22,  14,  55,  96,  38,  16,  74,  15,  11,  62,  94,  15,  16,
         62,  18,  44,  97,  38,  16,  98,  99,  16,  16, 100,  15,  16,  63,
         44, 101,  21,  22,  23,   7,  64,  13,  22,  14,  42,  16,  63,  42,
         11,  62,  18,   4,  16,  65])] Label: [0]
```

```
STR,"R3, [SP, #+0x04]"
LDR,"R2, [SP, #+0x78]"
ADD,"R1, SP, #8"
STRH,"R2, [SP, #+0x16]"
STR,"R1, [SP, #+0x1C]"
STR,"R1, [SP, #+0x10]"
MOVW,"R2, #0xFFFF"
SUBS,"R1, #1"
BEQ,#+0x30
ADD,"R3, SP, #124"
STR,"R0, [SP, #+0x18]"
STR,"R0, [SP, #+0x08]"
STRH,"R3, [SP, #+0x14]"
MOV,"R3, #0x208"
BLT,#+0x6A
SUB,"SP, SP, #112"
LDR,"R4, [R3]"
CMP,"R1, #0"
LDR,"R3, [PC, #+0x7C]"
PUSH,"{R4,LR}"
PUSH,{R2-R3}
BL,#-0x19F0
MOV,"R0, R3"
MOVW,"R1, #0x1392"
SUBS,"R3, #4"
ADD,"R3, R7, #0x13A0"
SUBS,"R2, #56"
```

Probabilities then class

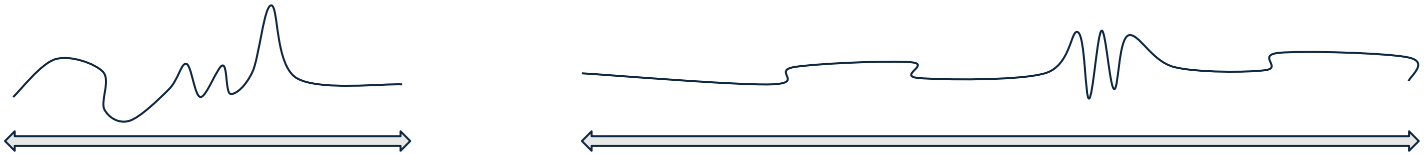
The model inside the model



Our results on function traces

Problems we had

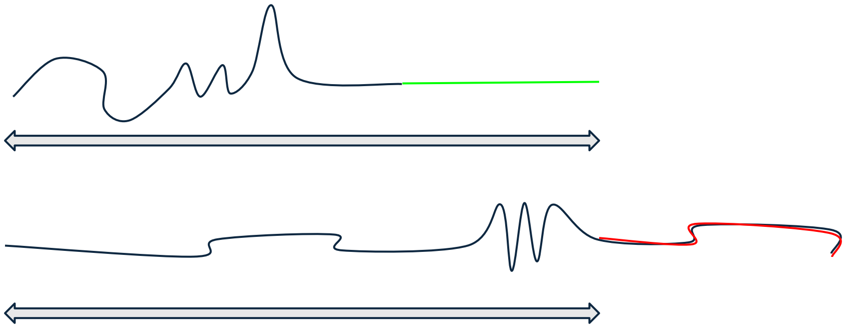
Model recognizes functions only with their length



malloc

memset

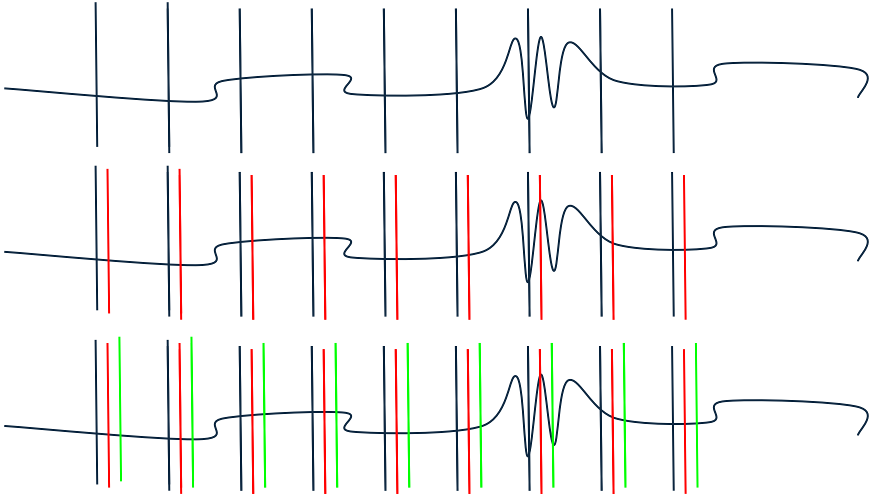
So we chose a standard length and padded or cropped the traces



We lost a big part of the dataset, especially on long traces, the model couldn't learn properly

Problems we had

We transformed the dataset in fixed length windows of traces

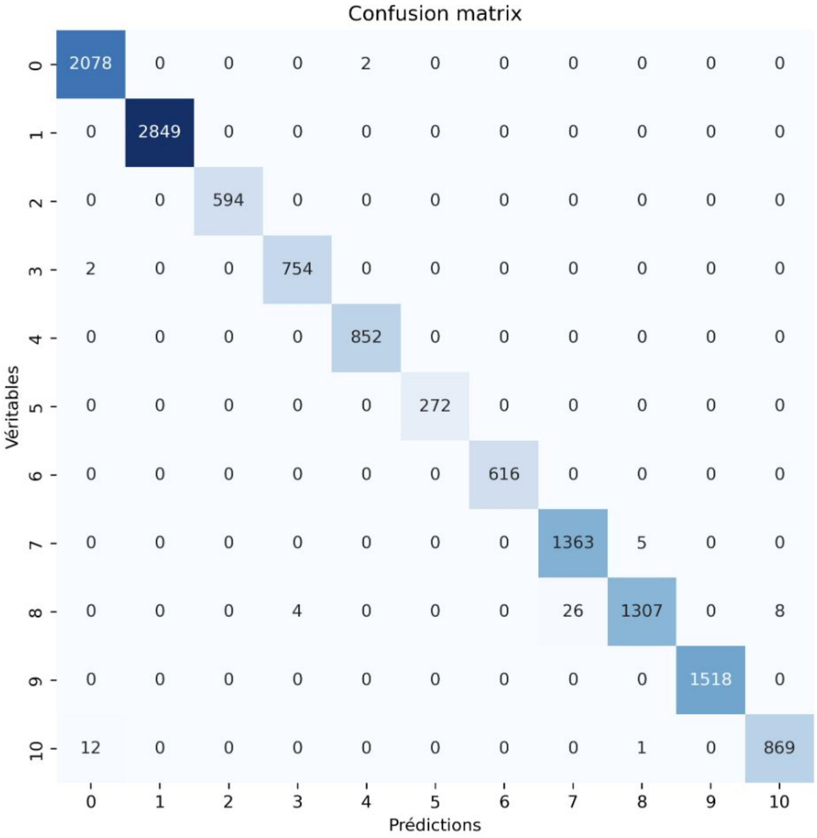


- Step-size of 1
- After different tries we chose windows of 100 timesteps
- Gave up on detecting the smallest function (free: 34 timesteps)

Unbalanced dataset : 1 malloc trace = 2 windows vs 1 snprintf trace = 250 windows
We chose a fixed number of window per class
We train the model to classify windows

Confusion matrix

acc = 99.98%



Dataset composition

Fonction	Entraînement	Test
`calloc`	2160	540
`free`	2160	540
`malloc`	2160	540
`memcopy`	2160	540
`memset`	2160	540
`memmove`	2160	540
`snprintf`	2160	540
`sprintf`	2160	540
`strcpy`	2160	540
`strcat`	2160	540
`strncpy`	2160	540
`strncat`	2160	540

Results

Confusion matrix - 12 fonctions

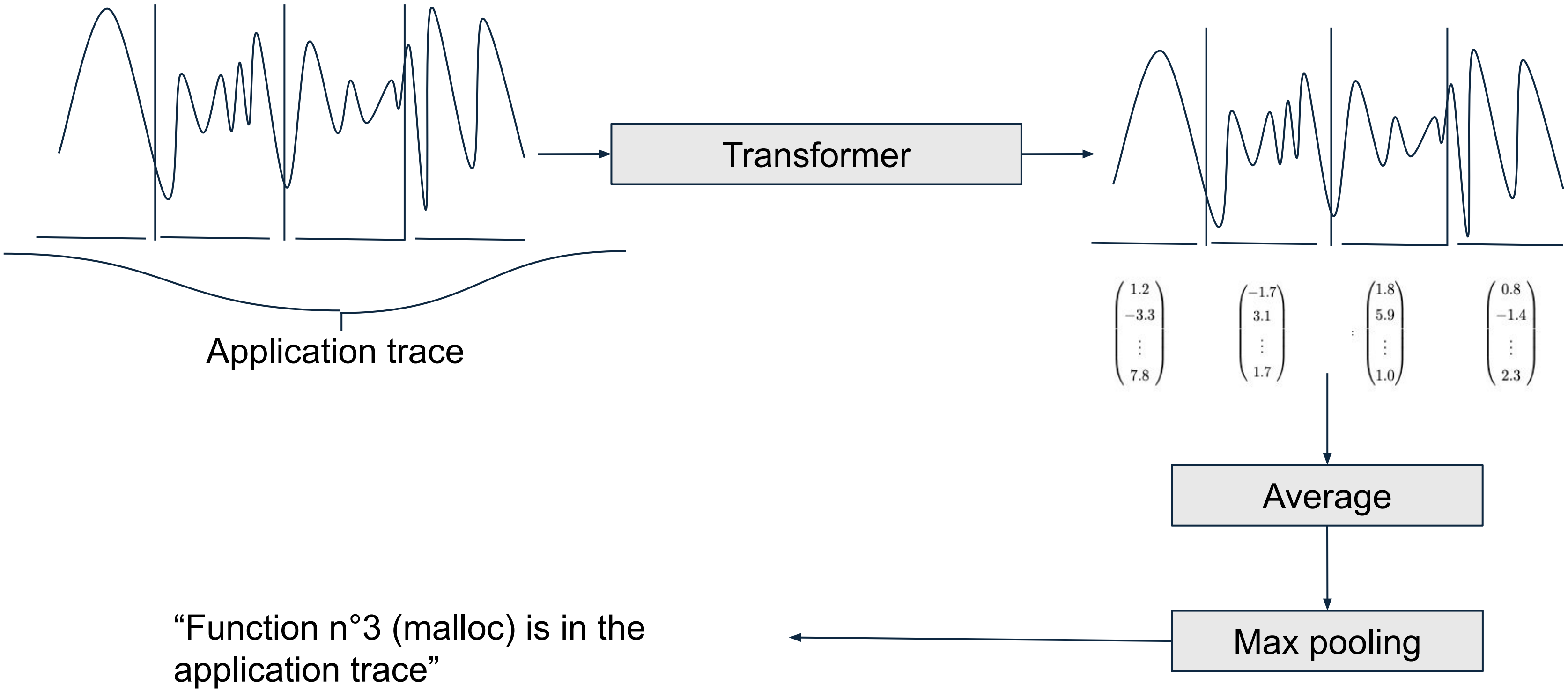
True label	Predicted label											
	calloc	free	malloc	memcopy	memmove	memset	snprintf	sprintf	strcat	strcpy	strncat	strncpy
calloc	540	0	0	0	0	0	0	0	0	0	0	0
free	0	540	0	0	0	0	0	0	0	0	0	0
malloc	0	0	540	0	0	0	0	0	0	0	0	0
memcopy	0	0	0	540	0	0	0	0	0	0	0	0
memmove	0	0	0	0	540	0	0	0	0	0	0	0
memset	39	0	0	0	0	501	0	0	0	0	0	0
snprintf	0	0	0	0	3	0	537	0	0	0	0	0
sprintf	0	0	0	0	0	0	0	540	0	0	0	0
strcat	0	0	0	0	0	0	0	0	539	1	0	0
strcpy	0	0	0	0	0	0	0	0	0	540	0	0
strncat	0	0	0	0	0	0	0	0	0	0	540	0
strncpy	0	0	0	0	0	0	0	0	0	0	0	540

Extension to application traces

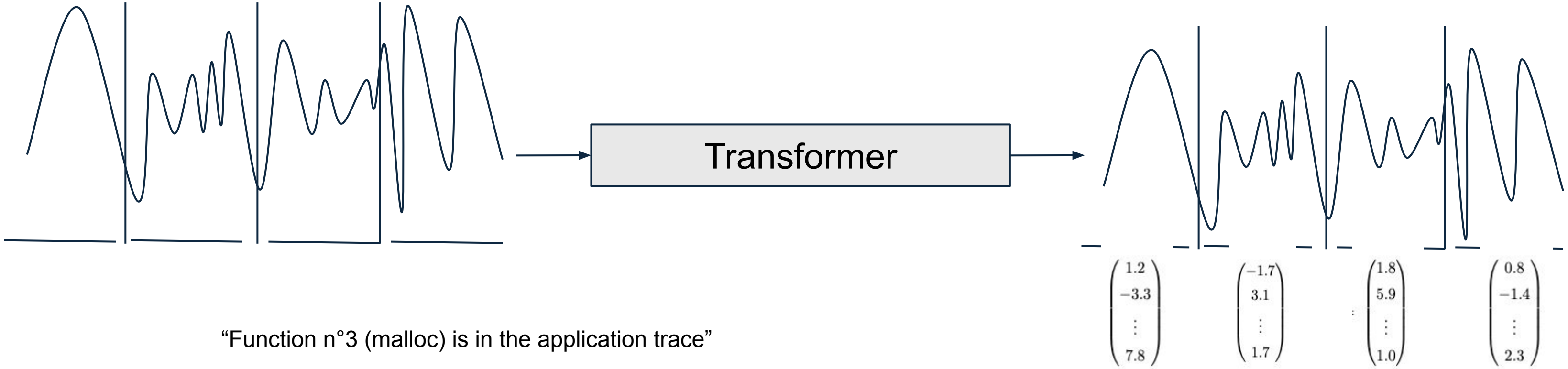
Functions detection in applications power trace

Goal : We want the model to recognize functions in application power traces

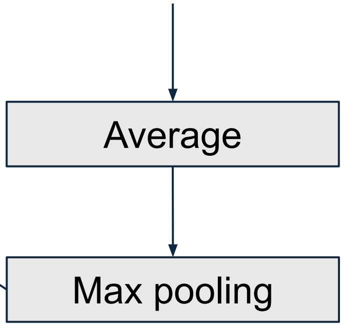
First method : Choose the prediction with the best average score



First method : Choose the prediction with best average score



“Function n°3 (malloc) is in the application trace”

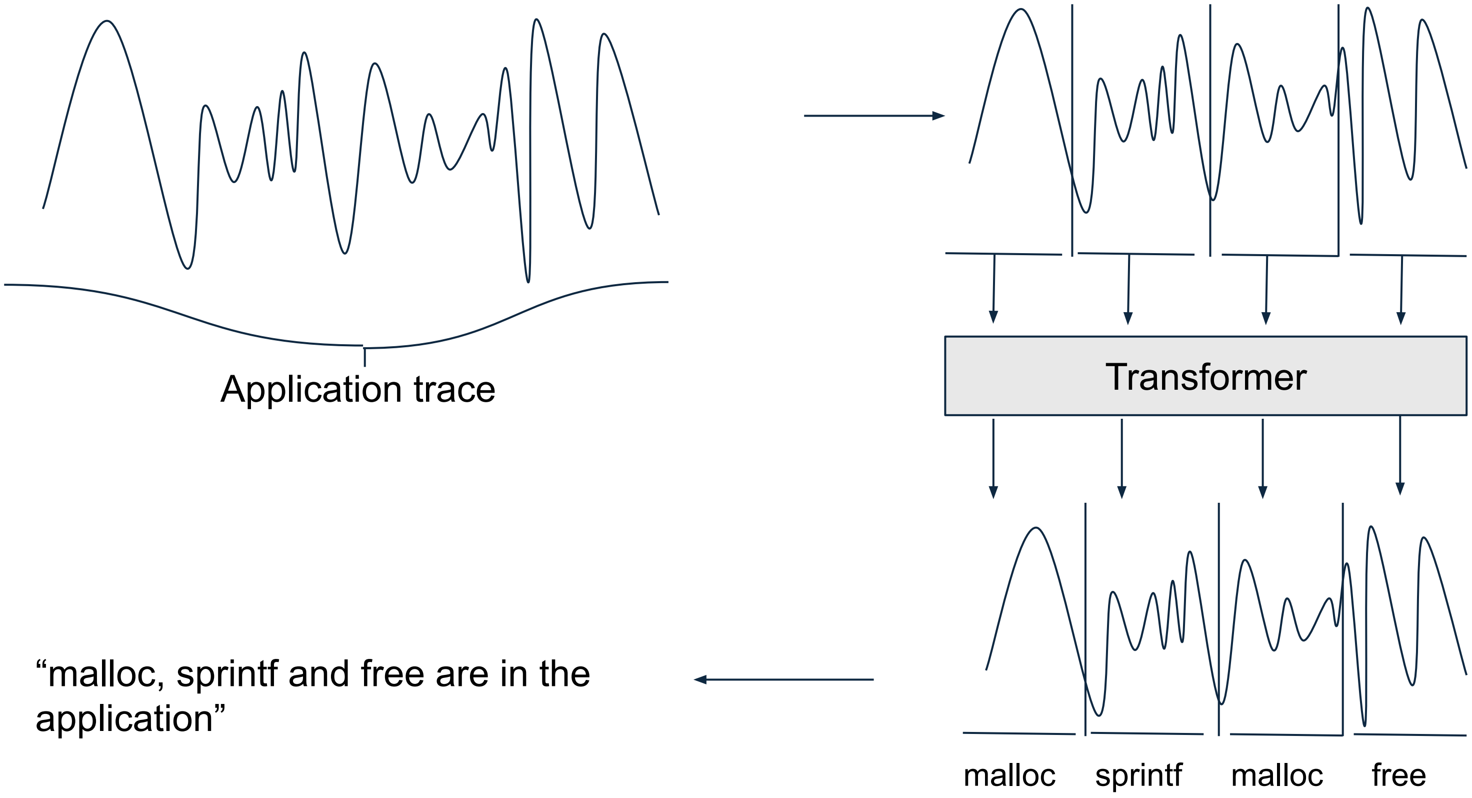


Problems : - Works only with applications that contain one function

- Functions that are not in the applications may have a good average score because they “look like” function that are
- Function that are really short compared to the app. trace may not be detected

Functions detection in applications power trace

Second method : Make a prediction for each window independently



Results for application sprcat (sprintf and strcat) with Method 1

```
Trace 97: fonction prédite = 'strncat'
Trace 97: fonction prédite = 'strncat'
Trace 97: fonction prédite = 'sprintf'
Trace 97: fonction prédite = 'strncat'
Trace 97: fonction prédite = 'strncat'
Trace 97: fonction prédite = 'strncat'
Trace 97: fonction prédite = 'strncat'
Trace 97: fonction prédite = 'strncat'
```

Prediction for one window

Prediction on applications

Application	1	2	3	4	5	6	7	8
Prédiction								

Final results

For power traces:

- Function power traces classification : almost 100%
- Function power traces identification in application traces : a few good results

For instruction traces:

- With only the identified function : almost 100%
- In a real context : can predict some functions and when there is nothing

Limitation of our work

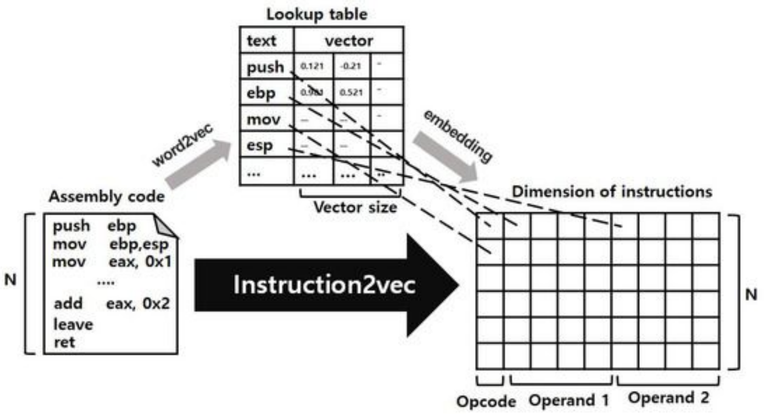
Limitations

- Training time can be long
- Creation of the vocabulary is delegated to a model made for human language
- Generalization is harder due to noise, shift and not having the knowledge of start and end points
- Defining the window seems easy during training but harder in reality

Next steps

Reviewing the window and making sure each sequences has already been seen

Implementing a new tokenization, more efficient based on paper : “Instruction2vec: Efficient Preprocessor of Assembly Code to Detect Software Weakness with CNN”



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

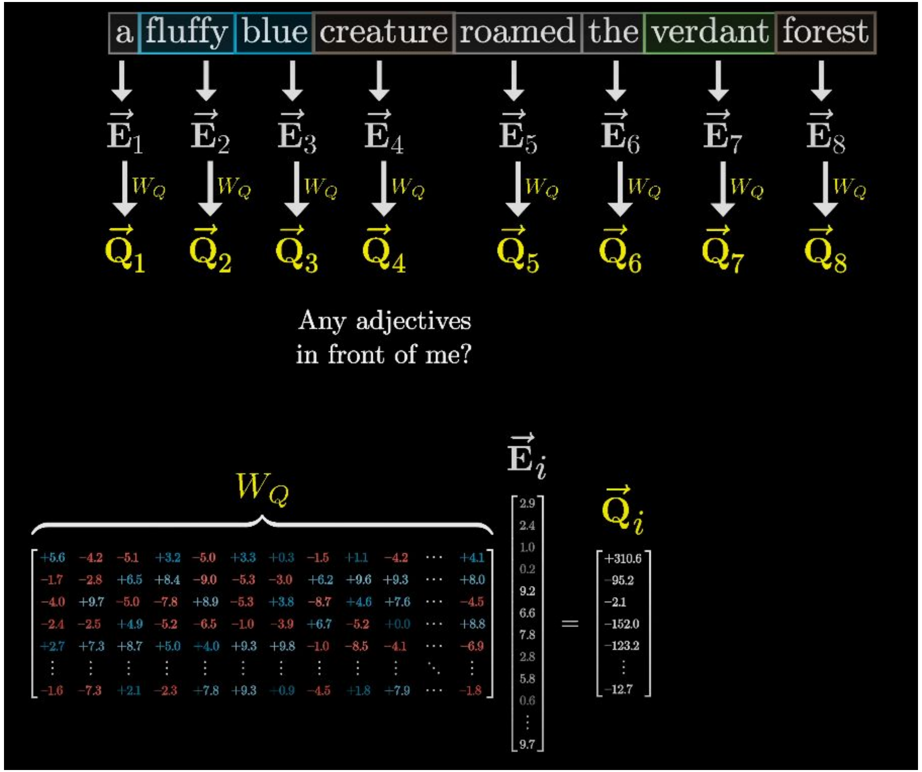
Thank you for your attention

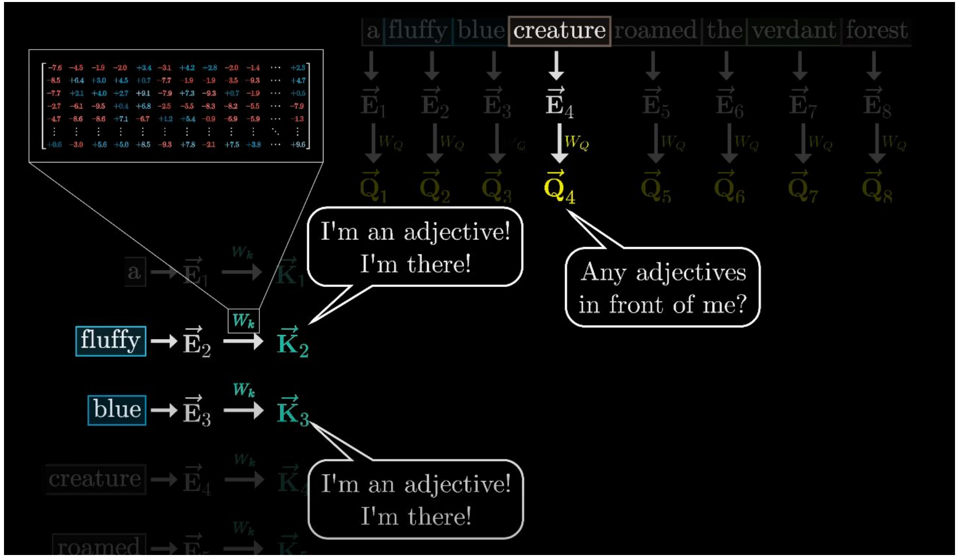
Timothée Léveillé
Auguste Célérier

Logo Partenaire

Annexes

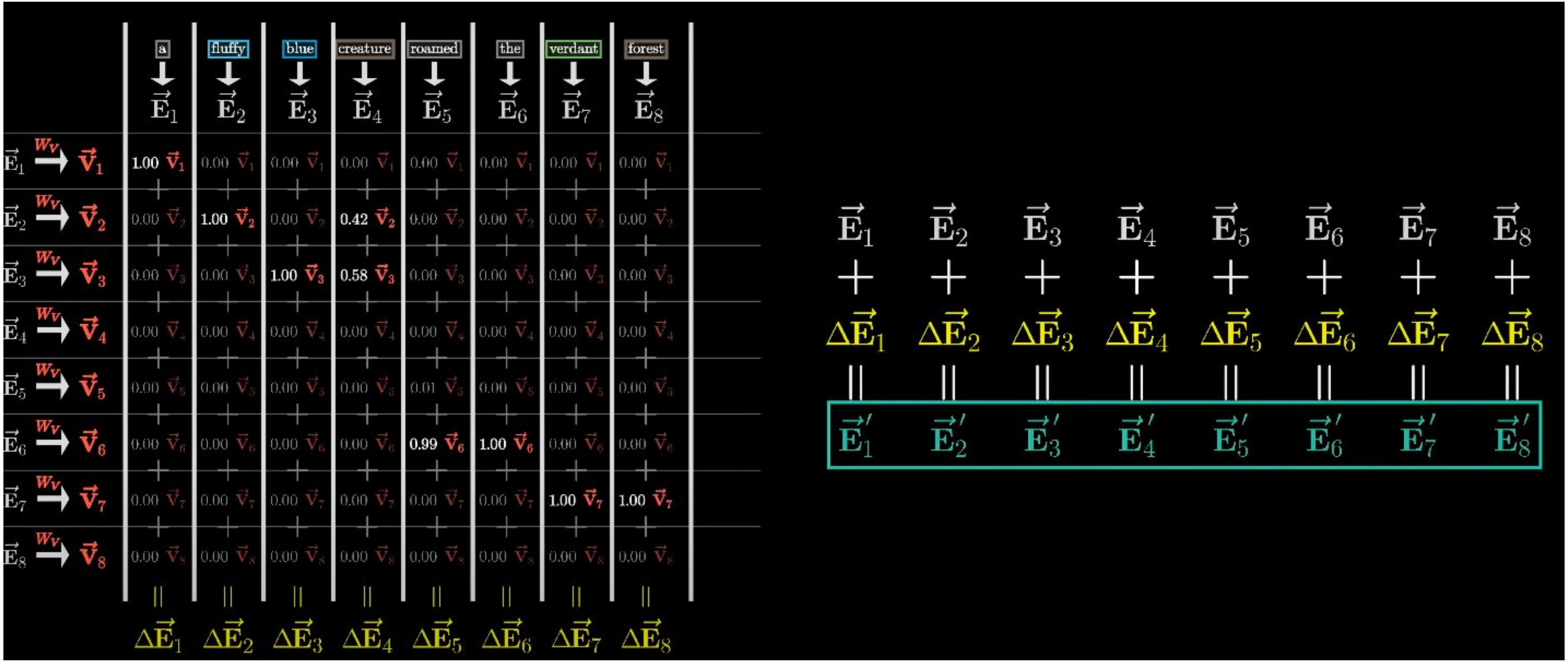
Query and key vectors





Query and key vectors

Value vectors and update of embeddings



Attention pooling and classifier

Attention pooling

For each vector we calculate an importance score

We softmax these scores

We multiply each vector by its softmaxed score and we sum

Classifier

z : output of attention pooling

W : matrice of trained weights to classify

b : trained bias

$$\text{logits} = Wz + b$$