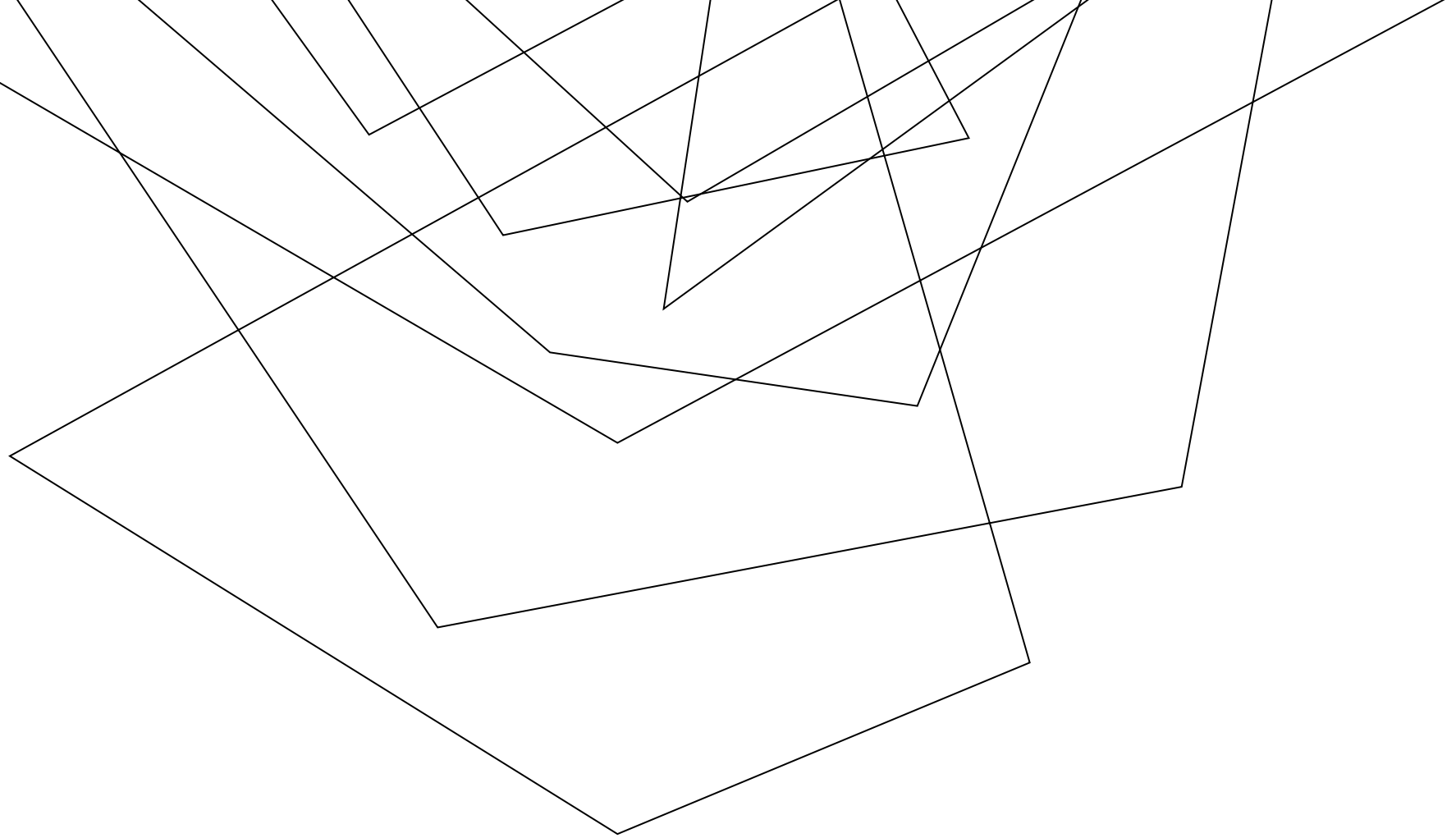
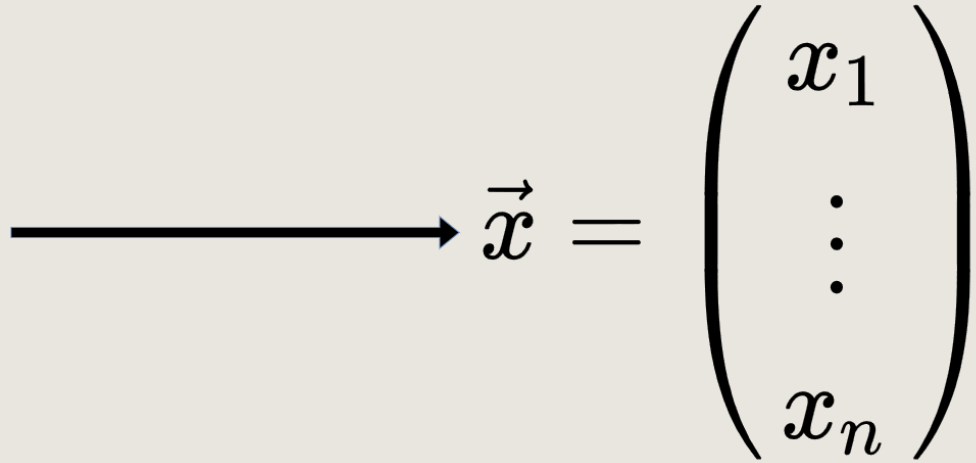


PITCH VORTRAG

Ruben Triwari

ASSEMBLY CODE EMBEDDING

```
1. square(int):
2.         push    rbp
3.         mov     rbp, rsp
4.         mov     DWORD PTR [rbp-4], edi
5.         mov     eax, DWORD PTR [rbp-4]
6.         imul    eax, eax
7.        pop     rbp
8.         ret
```



$$\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

- Nearest Neighbor Search
- Classification with Labels (e.g. Networking, time, ...)
- XFL: eXtreme Function Labeling
- Binary Code Similarity Detection (BCSD)

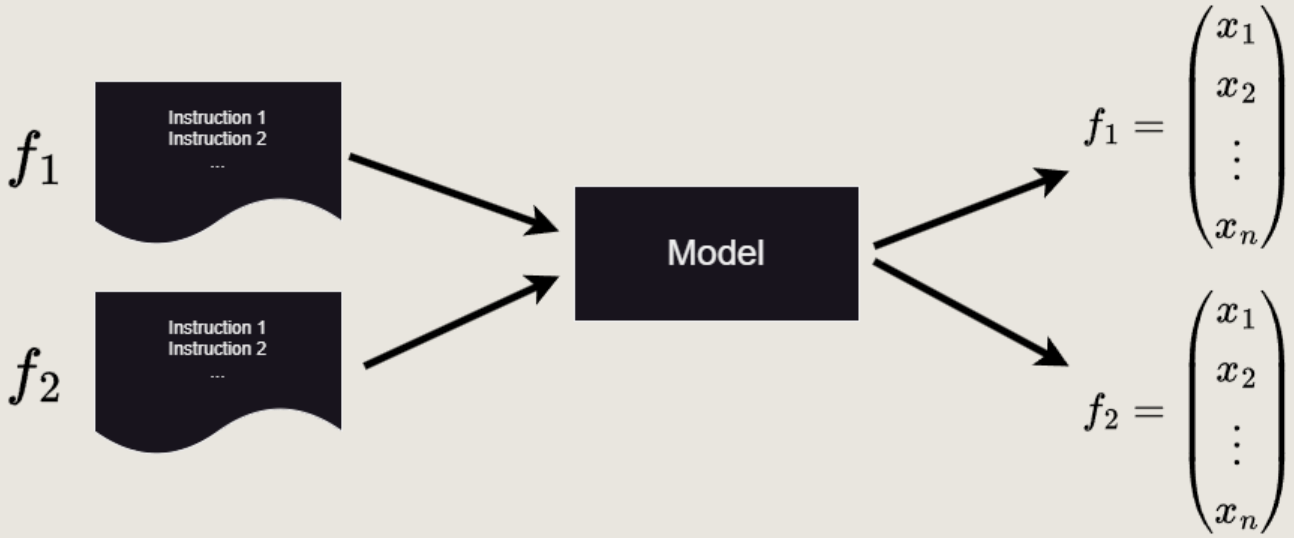
⇝ Assist Reverse Engineering

OVERVIEW OF DIFFRENT APPROACHES

Problem: What is a “good“ Assembly Embedding?

- SAFE: Same source code embeddings are close to each other
- PalmTree & JTrans: NLP Model Bert to learn assembly embeddings

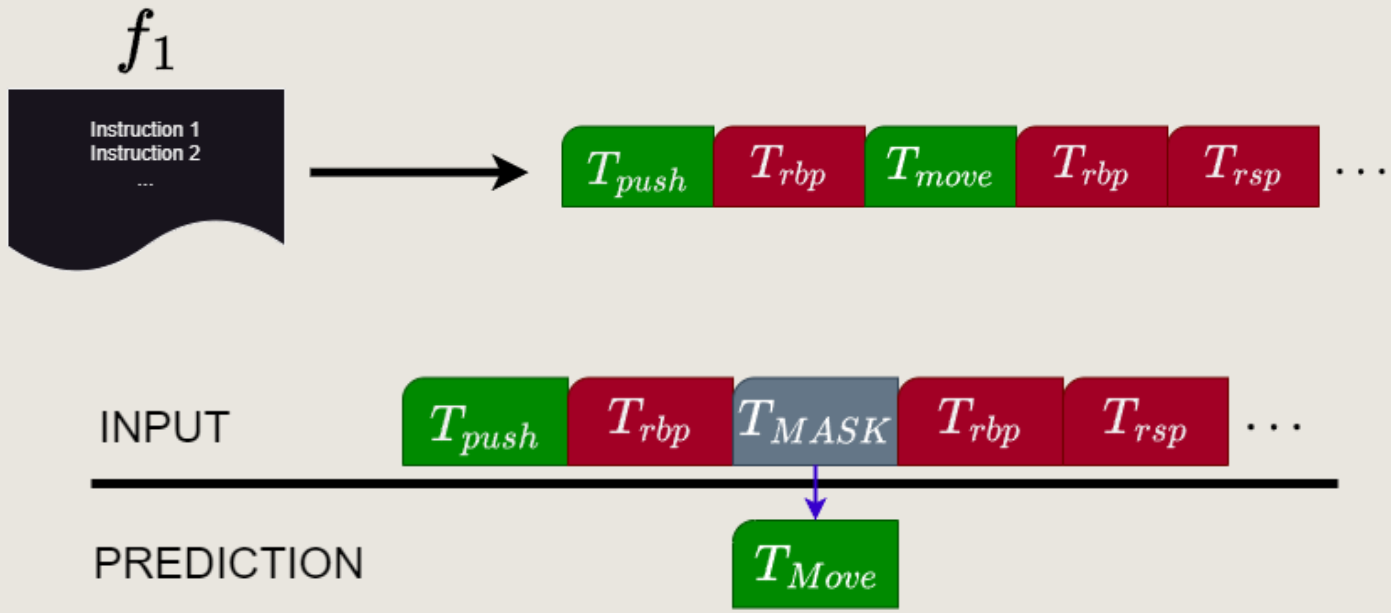
SAFE: SELF-ATTENTIVE FUNCTION EMBEDDINGS



- f_1 and f_2 same source code ⇝ minimize distance
- f_1 and f_2 different source code ⇝ maximize distance

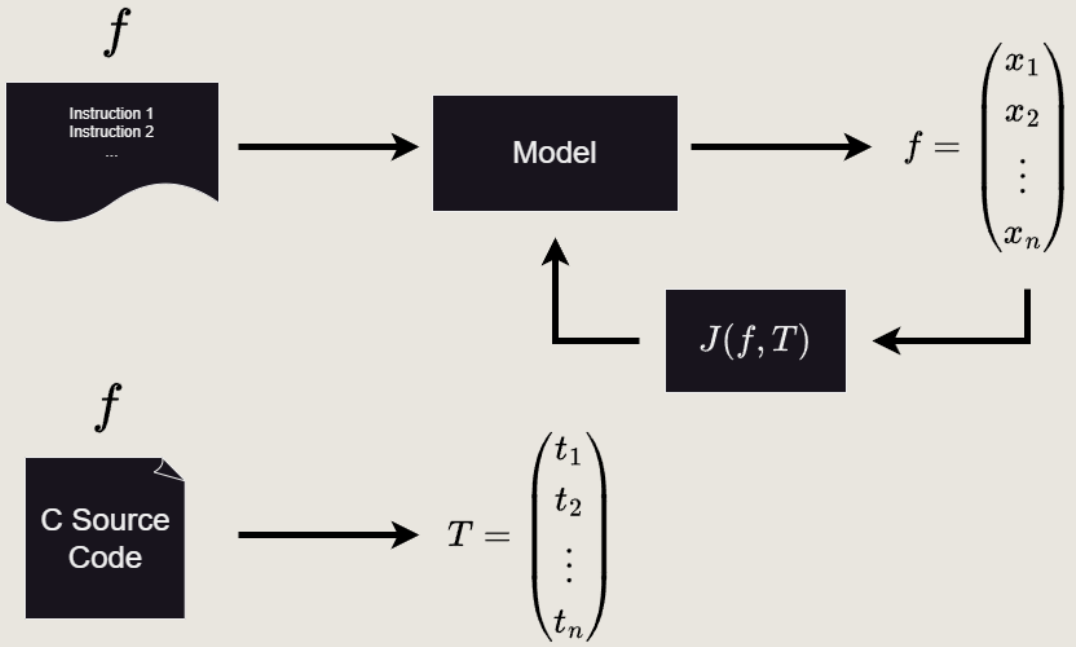
⇝ Similar functions are potentially far away from each other

PALMTREE & JTRANS



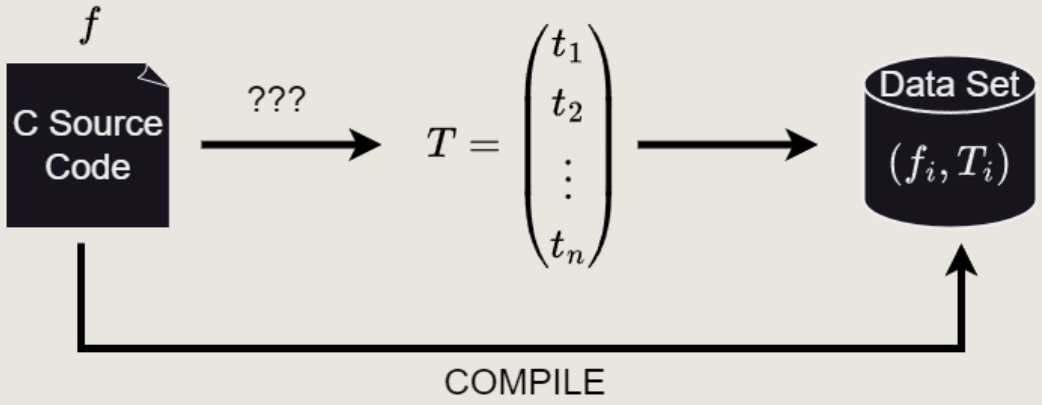
- Good semantic understanding of one function
- Bad understanding of how similar a function is to another

OUR APPROACH



- Supervised Learning
- Using C Source Code to generate “ground truth”

MY PART



- Sentence Transformer on C Source Code Comments
- Sentence Transformer on C Source Code Names
- Sentence Transformer on Llama generated Code Summaries
- Code2Vec

C SOURCE CODE COMMENTS

C SOURCE CODE NAMES

LLAMA GENERATED CODE SUMMARIES

EVALUATION

INTUITIVE ANSÄTZE FÜR TRAININGS DATEN

- Byte Darstellung des Assemblercodes
- Kontrollflussgraphen des Assemblercodes (Node == Basic Block)
- Darstellung als Reihen von Instruktionen des Assemblercodes

→ Semantik des Codes nur kaum oder gar nicht in den Trainingsdaten
→ Beim kompilieren zum Assemblercode gehen sehr viele Informationen Verloren

TRAININGS DATEN MIT SOURCE CODE INFORMATION