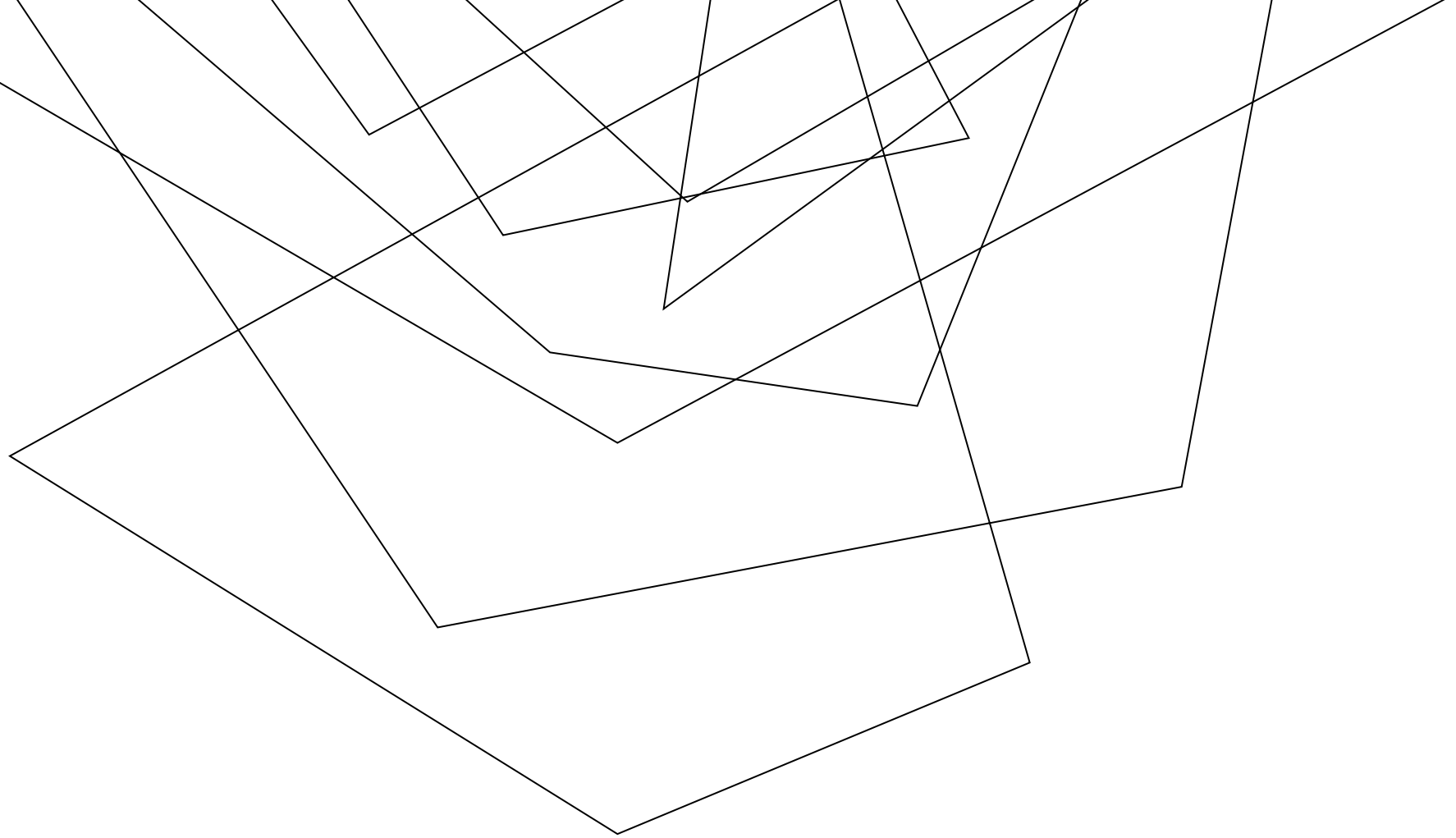
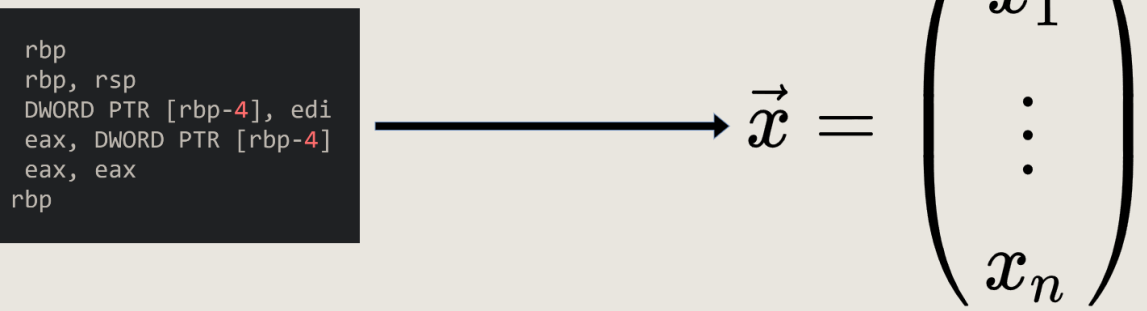


PITCH VORTRAG

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
```



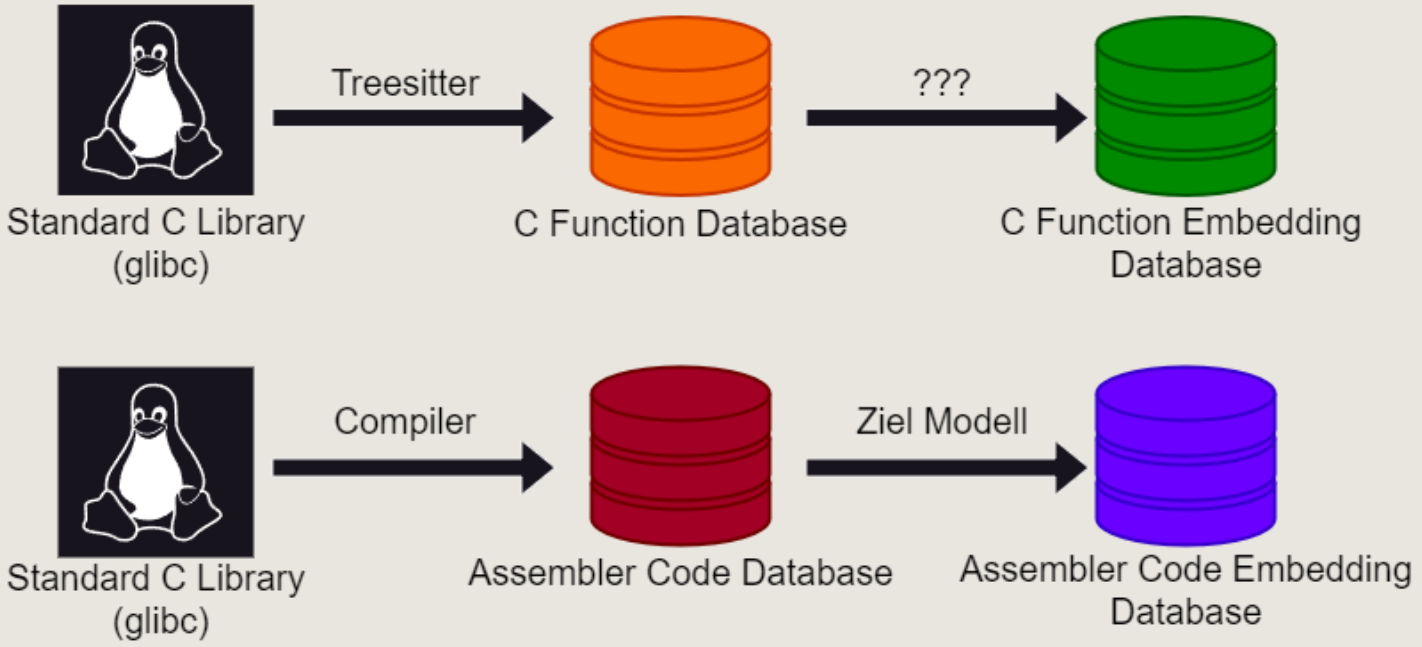
- Function Boundary Detection
- Binary Code Search
- Function Type Inference
  - ⇝ MaleWare Classifictaion
  - ⇝ Reverse Engineering

INTUITIVE ANSÄTZE FÜR TRAININGS DATEN

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

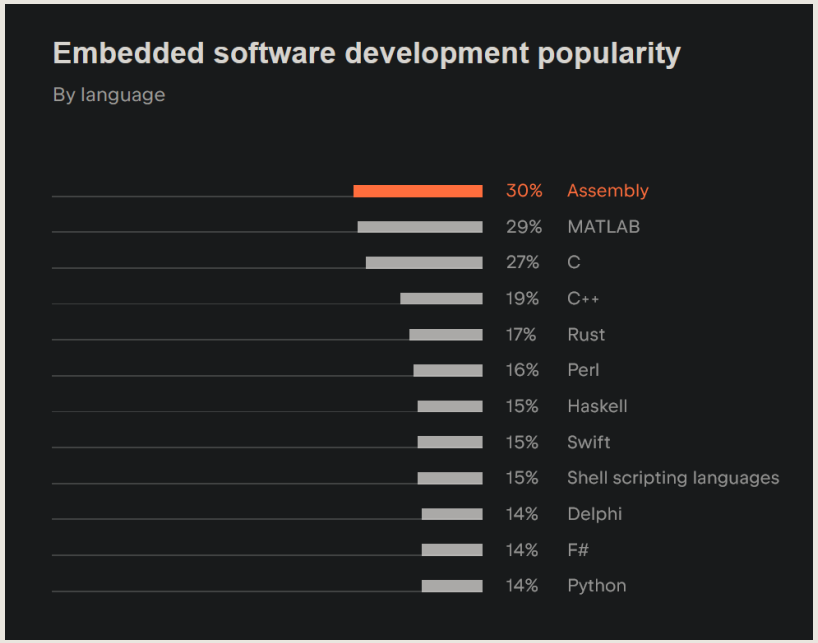
⇝ Semantik des Codes nur kaum oder gar nicht in den Trainingsdaten
⇝ Beim kompilieren zum Assemblercode gehen sehr viele Informationen Verloren

TRAININGS DATEN MIT SOURCE CODE INFORMATION



- C Function Embeddings erhalten Semantik und Struktur
  - ⇝ Ziel Modell nähert Embeddings an C Function Embeddings an
  - ⇝ Problem: Wie erzeugt man „gute“ C Function Embeddings?

WARUM C?



- Nah an Assembler Code
- Wird in nahezu jedem Kernel verwendet
- Wird in sehr vielen Geräten benützt (Eingebettete Systeme)
  - ⇝ Kritische Systeme verwenden C
  - ⇝ Bugs ohne Source Code zu patchen hat hohe Priorität

C EMBEDDINGS MIT NATÜRLICHER SPRACHE