Compiler and optimization level recognition using graph neural networks

Tristan Benoit
Université de Lorraine, CNRS,
LORIA,
F-54000 Nancy, France
tristan.benoit@loria.fr

Jean-Yves Marion
Université de Lorraine, CNRS,
LORIA,
F-54000 Nancy, France
jean-yves.marion@loria.fr

Sébastien Bardin
Université Paris-Saclay
CEA, LIST
Saclay, France
sebastien.bardin@cea.fr

Signatures

- A vulnerability is a weakness which can be exploited to perform unauthorized actions within a computer system.

Name	Description
CVE-2019-9924	rbash in Bash before 4.4-beta2 did not prevent the shell user from modifying BASH_CMDS, thus allowing the user to execute any command with the permissions of the shell.
CVE-2019-9146	Jamf Self Service 10.9.0 allows man-in-the-middle attackers to obtain a root shell by leveraging the "publish Bash shell scripts" feature to insert "/Applications/Utilities/Terminal app/Contents/MacOS/Terminal" into the TCP data stream.
CVE-2019-1596	A vulnerability in the Bash shell implementation for Cisco NX-OS Software could allow an authenticated, local attacker to escalate their privilege level to root. The attacker must authenticate with valid user credentials. The vulnerability is due to incorrect permissions of a system executable. An attacker could exploit this vulnerability by authenticating to the device and entering a crafted command at the Bash prompt. A successful exploit could allow the attacker to escalate their privilege level to root. Nexus 3000 Series Switches are affected in versions prior to 7.0(3)I7(4). Nexus 3500 Platform Switches are affected in versions prior to 7.0(3)I7(4). Nexus 3600 Platform Switches are affected in versions prior to 7.0(3)F3(5). Nexus 9000 Series Switches in Standalone NX-OS Mode are affected in versions prior to 7.0(3)I7(4). Nexus 9500 R-Series Line Cards and Fabric Modules are affected in versions prior to 7.0(3)F3(5).
CVE-2019-1593	A vulnerability in the Bash shell implementation for Cisco NX-OS Software could allow an authenticated, local attacker to escalate their privilege level by executing commands authorized to other user roles. The attacker must authenticate with valid user credentials. The vulnerability is due to the incorrect implementation of a Bash shell command that allows role-based access control (RBAC) to be bypassed. An attacker could exploit this vulnerability by authenticating to the device and entering a crafted command at the Bash prompt. A successful exploit could allow the attacker to escalate their privilege level by executing commands that should be restricted to other roles. For example, a dev-ops user could escalate their privilege level to admin with a successful exploit of this vulnerability.
CVE-2018-7739	antsle antman before 0.9.1a allows remote attackers to bypass authentication via invalid characters in the username and password parameters, as demonstrated by a username=>&password=%0a string to the /login URI. This allows obtaining root permissions within the web management console, because the login process uses Java's ProcessBuilder class and a bash script called antsle-auth with insufficient input validation.

Signatures

- Library functions identification resorts on signatures.

```
.text:004069F1  neg     ecx
.text:004069F3  push    edx
.text:004069F4  lea     edx, [esp+2Ch+var_C]
.text:004069F8  sbb     ecx, ecx
.text:004069FA  push    0
.text:004069FC  and     ecx, edx
```

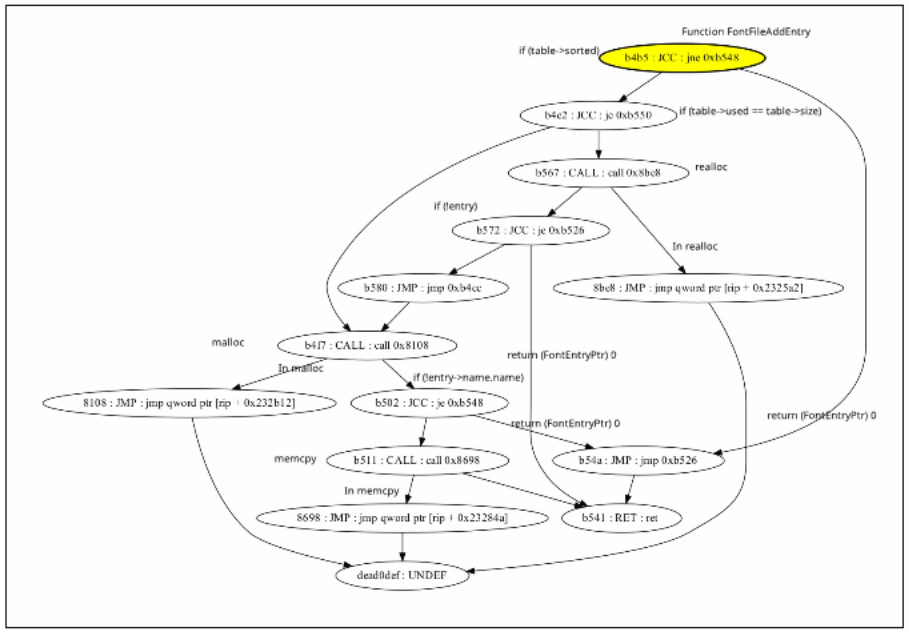
```
.text:00406A02  neg     edi
.text:00406A04  push    ecx
.text:00406A05  lea     ecx, [esp+34h+var_10]
.text:00406A09  sbb     edi, edi
.text:00406A0B  and     edi, ecx
```

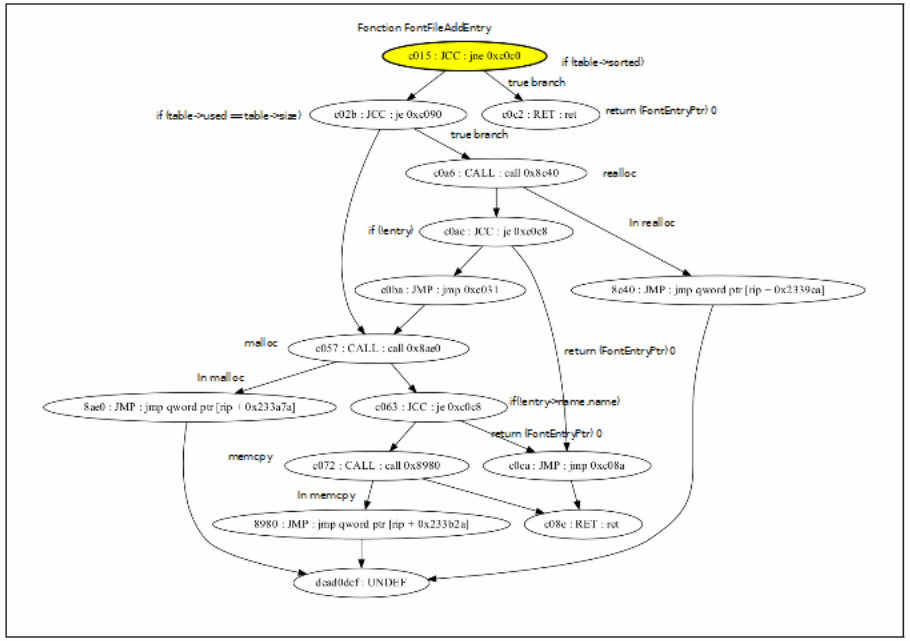
```
.text:00406A11  neg     esi
.text:00406A13  sbb     esi, esi
.text:00406A15  push    edi
.text:00406A16  and     esi, edx
```

Qiu, J. et al. "Library functions identification in binary code by using graph isomorphism testings." 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER) (2015): 261-270.

Signatures

- Compiling environment and compiler options modify the shape of signatures.

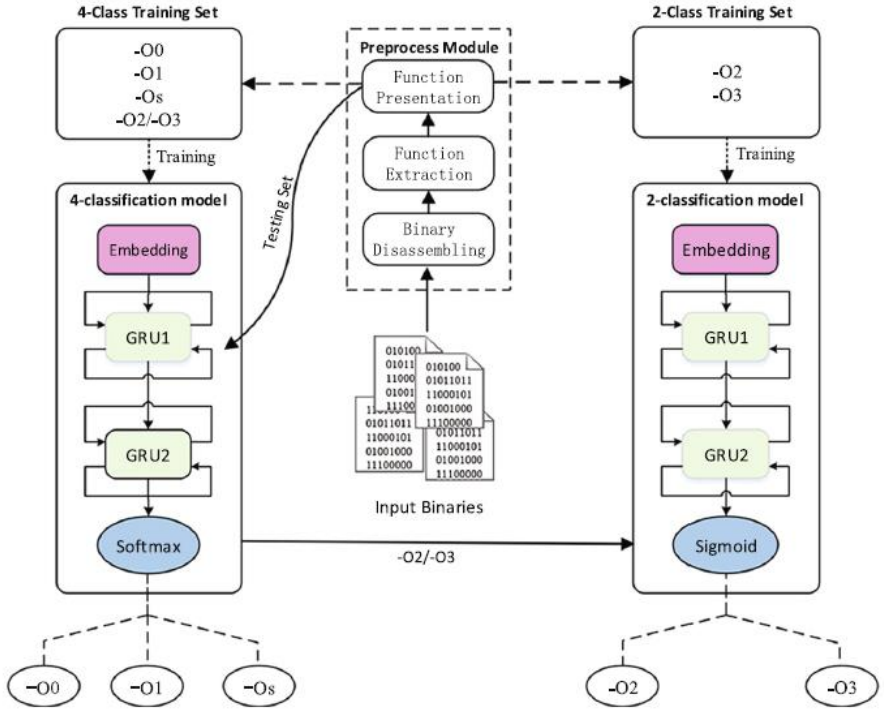




Goal

- Identifying the **toolchain provenance**, i.e. the compiler family (e.g. Visual Studio), the compiler version (e.g. 10.0, 12.0) and its optimization options (e.g. -O1, -O2), that have been used to produce a **stripped binary code**.

Related works



Compiler Family $\mathcal{C}$	Version $\mathcal{V}$	Optimization Level $\mathcal{O}$ Low	High
GNU Compiler Collection (GCC)	3.4.x	-O0,-O1	-O2,-O3
	4.2.x	-O0,-O1	-O2,-O3
	4.3.x	-O0,-O1	-O2,-O3
	4.4.x	-O0,-O1	-O2,-O3
Intel Compilers (ICC)	10.x	-O0	-O2,-O3
	11.x	-O0	-O2,-O3
Microsoft Visual C++ (MSVC)	VS 2003	/Od	/O2
	VS 2005	/Od	/O2
	VS 2008	/Od	/O2

Chen at al. "Himalia: Recovering compiler optimization levels from binaries by deep learning", Intelligent Systems and Applications, 2019

Rosenblum et al. "Recovering the Toolchain Provenance of Binary Code", International Symposium on Software Testing and Analysis, 2011

Related works

- Focus on function output.
- Small amount of data for each function.

- Thousands of functions in one binary file.
- A rather limited number of binary file considered.

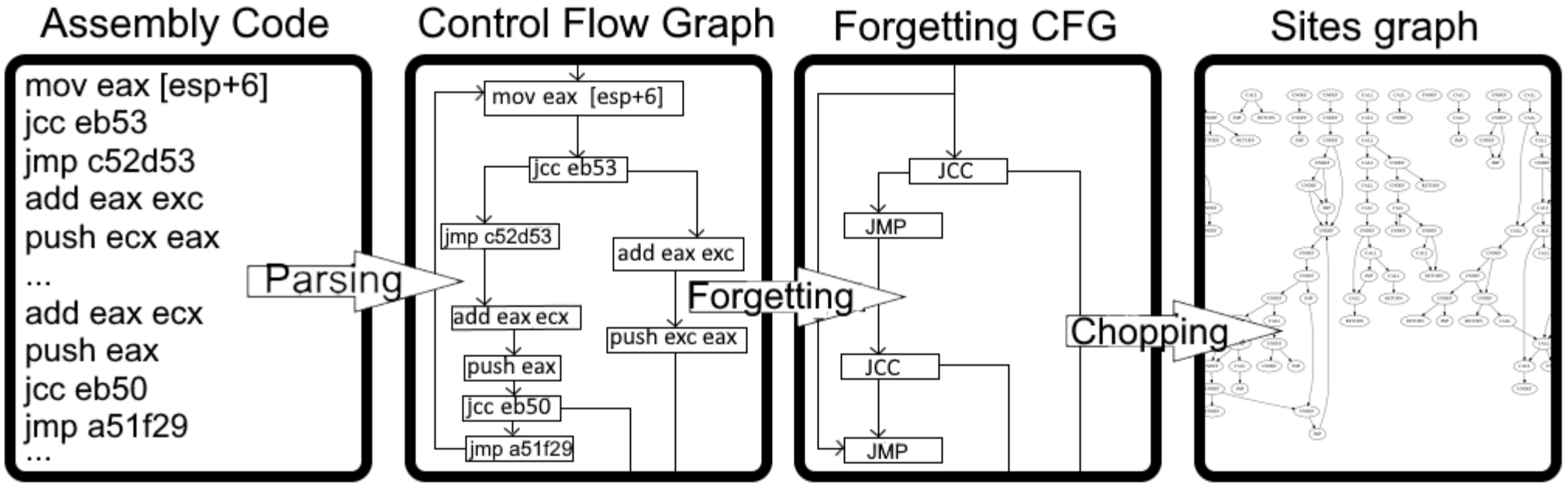
- Each source code is compiled with every possible toolchain configuration.

Contribution

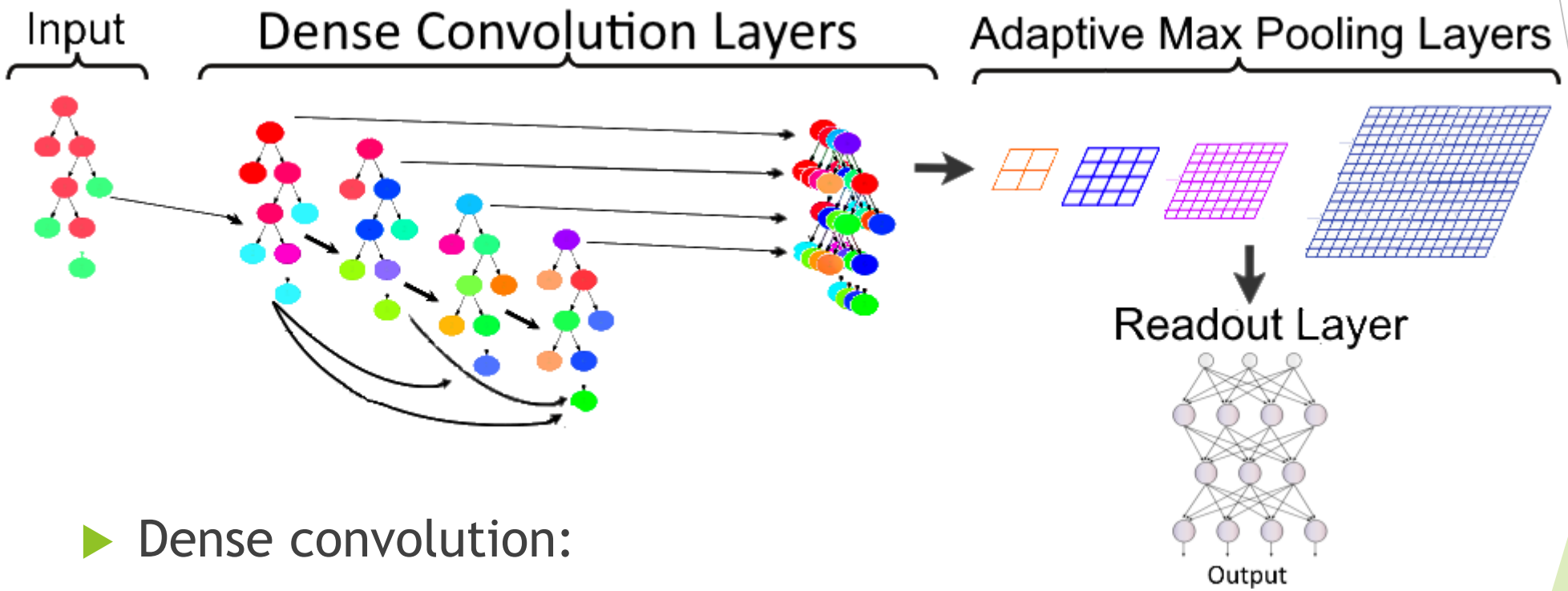
- A **Graph Neural Network** based framework to determine the **toolchain provenance** of a **whole stripped binary**.
- An evaluation of this process on a **broad** dataset composed from **36,272** source code compiled with **92** toolchain configurations.

Site Neural Network

- We only keep a skeletal part of the CFG.
- We extract 100 small graphs from it.



Site Neural Network



- Dense convolution:

$$Y_1 = (A + I)X_0W_0 + b_0)$$

$$(Y_{k+1})_{k\geq 0} = ((A + I)Y_kW_k + b_k)|Y_k)$$

Dataset

The dataset is made from 36,272 C/C++ source files solving 91 problems from Codeforces.

- Compiler family: Visual Studio, MinGW, Clang and GCC.
- Optimization level: O0, 01, O2, O3 and Os.
- Compiler version: from 5 to 6 for each family.

Research questions

What is the capacity of our framewok to predict:

- Compiler family?
- Optimization level?
- Compiler version?

Compiler family

Compiler	Precision	Recall	F1 Score	Support
Clang	1	0.9933	0.9967	600
GCC	0.9967	1	0.9983	600
MinGW	0.9983	0.9917	0.9950	600
VS	0.9828	0.9975	0.9901	400
Macro AVG	0.9944	0.9956	0.9950	2200

- Very good accuracy on this task.

Optimization level

Option	Precision	Recall	F1 Score	Support
-O0	0.7917	0.8261	0.8085	460
-O1	0.8289	0.7478	0.7863	460
-O2/-03	0.7869	0.8329	0.8092	820
-Os	0.6316	0.6	0.6154	460
Macro AVG	0.7598	0.7517	0.7549	2200

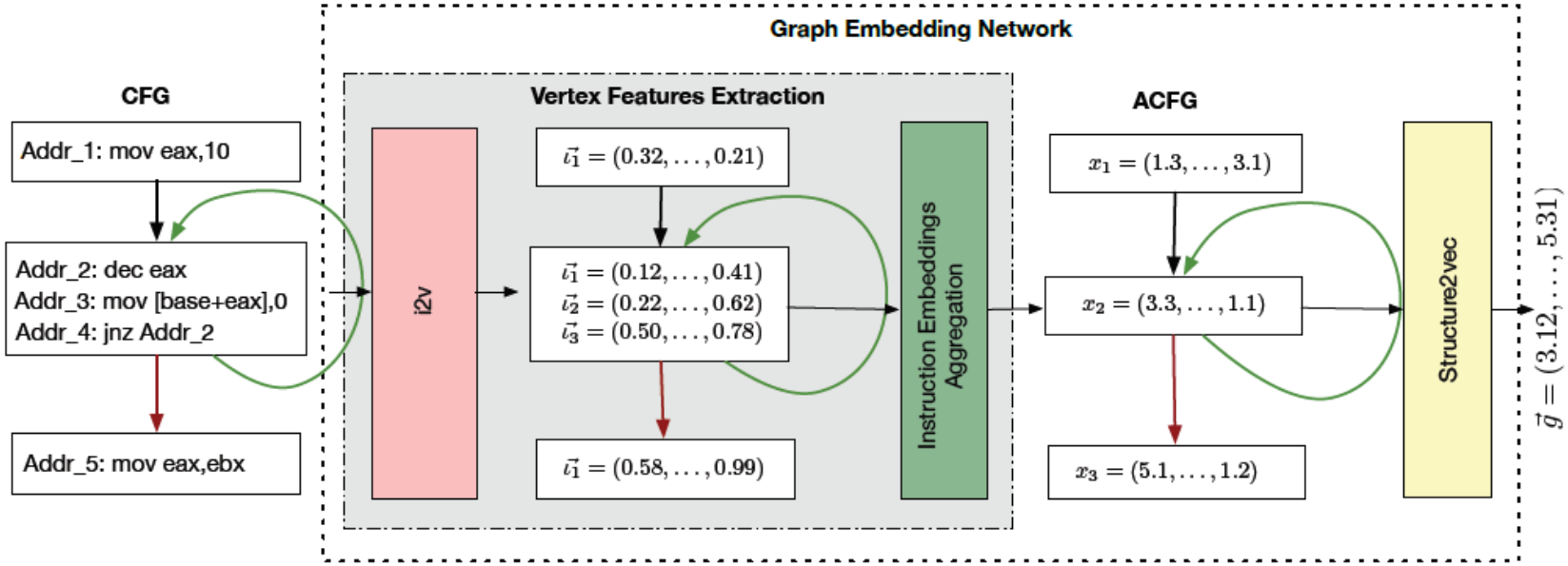
- Good accuracy on this task.

Compiler version

Family	Precision	Recall	F1-Score	Support
Clang	0.2019	0.1883	0.1856	600
GCC	0.5731	0.5567	0.5496	600
MinGW	0.9832	0.9767	0.9799	600
VS	0.9162	0.9275	0.9206	400
GCC-MinGW-VS	0.8242	0.8202	0.8167	1600
MinGW-VS	0.9497	0.9521	0.9502	1000
Macro AVG	0.6686	0.6623	0.6589	2200

- Very good accuracy on this task for families such as MinGW and Visual Studio but not Clang and GCC.

Comparison

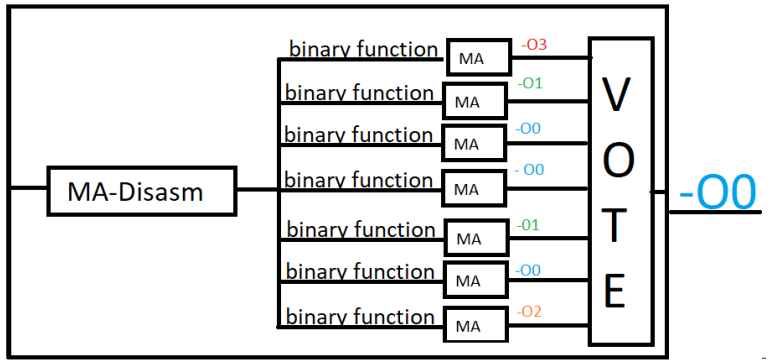


Massarelli et al. "Investigating graph embedding neural networks with unsupervised features extraction for binary analysis.", 2019.

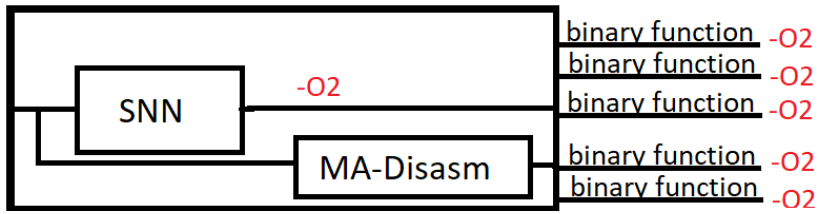
- How do we compare to [Massarelli et al., 2019] in terms of accuracy and performance?

Comparison

- We change the output level of Massarelli et al. framework and SNN.



MA-B outputs binary level prediction



SNN-F outputs function level prediction

Comparison

Framework / Task	Compiler	Optimization	Version
MA-B	0.91	0.42	0.32
SNN	0.92 [±0.03]	0.58 [±0.03]	0.45 [±0.02]
MA	**0.90**	0.36	0.36
SNN-F	0.87 [±0.07]	**0.60** [±0.05]	**0.42** [±0.02]

- We outperform Massarelli et al. framework at both the function and the binary level.
- Compared to Massarelli et al. framework, SNN is 68 x times faster during learning and 1300 x time faster during preprocessing.

Conclusion

- Classification at the whole binary level works.
- Our simple framework is efficient.

Direction for future works:

- Richer semantic features.
- Natural language processing techniques.
- Comparisons between frameworks.

Conclusion

- Thank you for your attention!