



# WebSVF



## Static Analysis Leveraging SVF and CodeGPT

UNSW | Software Engineering | T3 2025 | Christian Tolentino and Joshua Wills

Supervisor: Yulei Sui  
Assessor: Yuekang Li

# Overview

1. Recap
  - a. Problem Statement
  - b. General Context
  - c. Previous Features
2. Joshua's Improvements
  - a. C++ Support
  - b. Custom Keybindings
  - c. Default Code
3. Christian's Improvements
  - a. User Onboarding Guide
  - b. Terminal Feature Fix/Improvements
  - c. Quality of Life Improvements
4. Looking to the Future

# Problem Statement

There is a significant problem in modern software development relating to the identification of bugs and security risks in large, complicated applications. As the complexity of such applications grows, the difficulty of identifying such issues worsens. The goal of WebSVF, including CodeGPT, is to provide a simple interface for users to easily identify such bugs and security risks. This is achieved by leveraging SVF, the Static Value-Flow Analysis Framework, and modern LLMs to assist in the detection of vulnerabilities, generate graphs of code structure and explain blocks of software.

# General Context | Static Analysis

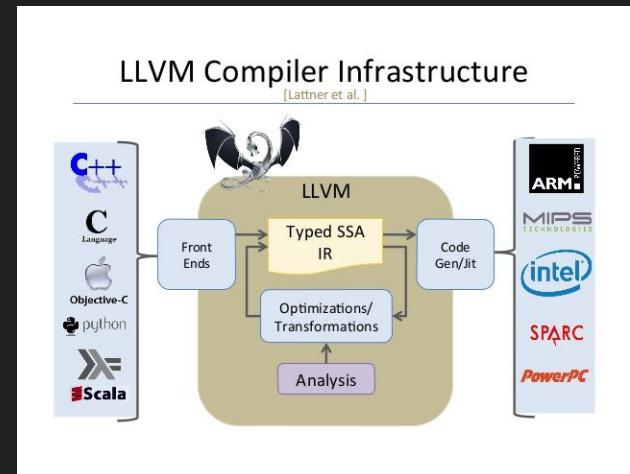
- The abstract interpretation of programs to gain insight on the actual computations of a program - Cousot 1970s
- Gain an understanding of a software project's internal structure
- Identify common programming errors
  - Buffer overflows
  - Memory leaks
  - Null pointer dereference
  - etc.



Patrick Cousot, a father of Static Analysis techniques

# General Context | LLVM

- Stands for ‘Low Level Virtual Machine’
  - Albeit a somewhat misleading name
  - First developed by Chris Lattner in the early 2000s
- In practice, a compiler framework, which features a low-level and platform-generic intermediate representation (IR)
  - IR is in a Single Static Assignment (SSA) form
- SVF operates on the bitcode of LLVM.
  - Multi language support!



LLVM operating on multiple compiler frontends

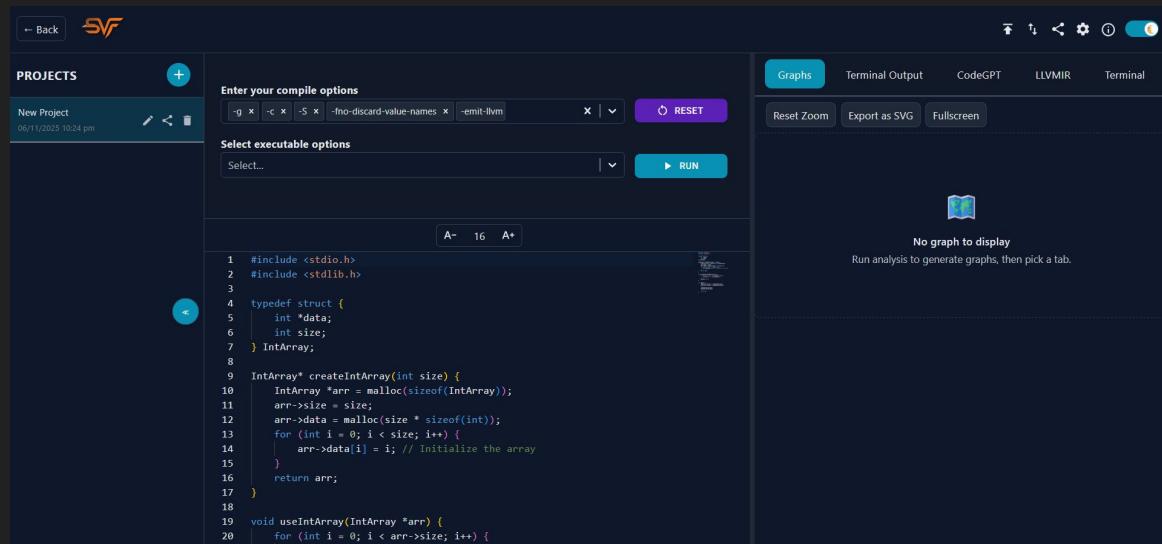
# Context | SVF

- ‘Static Value-Flow Analysis Framework’
- “Source code analysis tool that enables interprocedural dependence analysis for LLVM-based languages.”



# Context | WebSVF

- Web-based application for interfacing with SVF
- 6th major version
- Lowers barrier to entry
- Single paged



The screenshot shows the WebSVF interface. At the top, there's a navigation bar with a back button, a logo, and various settings. Below it is a 'PROJECTS' section with a 'New Project' button and a timestamp. The main area contains a code editor with the following C code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct {
5     int *data;
6     int size;
7 } IntArray;
8
9 IntArray* createIntArray(int size) {
10    IntArray *arr = malloc(sizeof(IntArray));
11    arr->size = size;
12    arr->data = malloc(size * sizeof(int));
13    for (int i = 0; i < size; i++) {
14        arr->data[i] = i; // Initialize the array
15    }
16    return arr;
17 }
18
19 void useIntArray(IntArray *arr) {
20    for (int i = 0; i < arr->size; i++) {
21        printf("%d\n", arr->data[i]);
22    }
23 }
```

Below the code editor are 'Enter your compile options' and 'Select executable options' fields. A 'RUN' button is located next to the executable options field. On the right side, there are tabs for 'Graphs' (which is selected), 'Terminal Output', 'CodeGPT', 'LLVMIR', and 'Terminal'. Below these tabs are buttons for 'Reset Zoom', 'Export as SVG', and 'Fullscreen'. A message at the bottom right says 'No graph to display. Run analysis to generate graphs, then pick a tab.'

A snapshot of WebSVF in its current state

# Context | WebSVF

Graphs      Terminal Output      CodeGPT      LLVMIR

A- 16 A+

```
1 ****CallGraph Stats*****
2 ###### (program : )#####
3
4 -----
5 CalRetPairInCycle 0
6 TotalFedge 2
7 MaxNodeInCycle 0
8 NodeInCycle 0
9 TotalCycle 0
10 TotalNode 4
11 #####
12 ****General Stats*****
13 ###### (program : )#####
14
15 AddrNum 12
16 TotalFieldObjects 0
17 StoresNum 5
18 SbitCastNumber 0
19 FSObjNum 9
20 BBWith3Succ 0
21 FIObjNum 0
22 LoadsNum 5
23 MaxStructSize 0
24 IndCallsites 0
25 GepsNum 1
26 BBWith2Succ 0
27 TotalIPTAPAGEDges 17
28 TotalPAGEdges 55
29 TotalPointers 38
30 ReturnsNum 0
31 VarArrayObj 0
```

Example of the terminal output

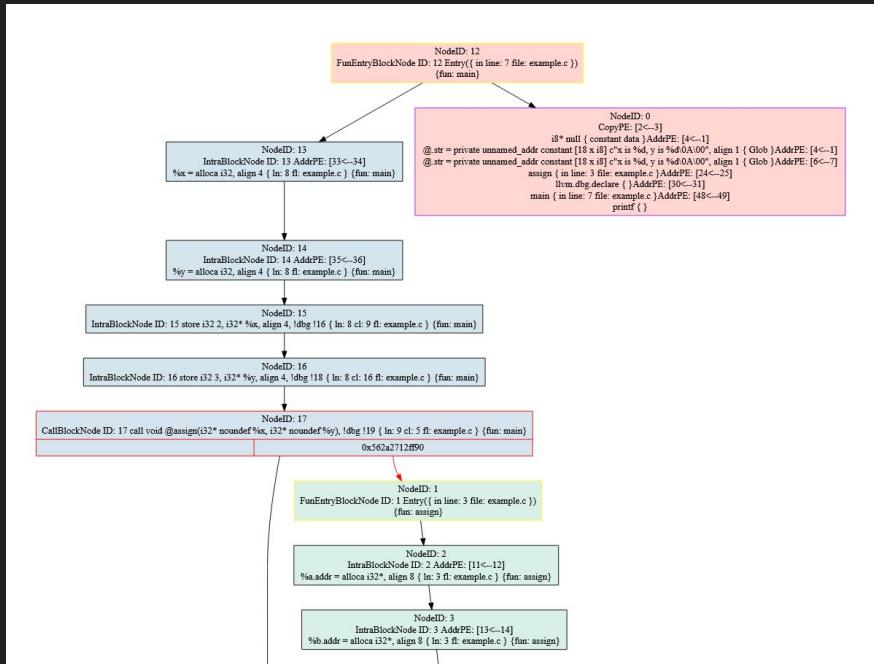
Graphs      Terminal Output      CodeGPT      LLVMIR

A- 12 A+

```
1 ; ModuleID = 'example.c'
2 source_filename = "example.c"
3 target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-164:64-f80:128-n8:16:32:64-S128"
4 target triple = "x86_64-pc-linux-gnu"
5
6 @.str = private unnamed_addr constant [18 x i8] c"x is %d, y is %d\bA\b0", align 1
7
8 ; Function Attrs: noinline nounwind optnone untable
9 define dso_local void @ssign(i32* noundef %a, i32* noundef %b) #0 !dbg !10 {
10   entry:
11     %a.addr = alloca i32*, align 8
12     %b.addr = alloca i32*, align 8
13     store i32 %a, i32** %a.addr, align 8
14     call void @_lvm.debug.declare(metadata i32** %a.addr, metadata !16, metadata !DIExpression()), !dbg !17
15     store i32 %b, i32** %b.addr, align 8
16     call void @_lvm.debug.declare(metadata i32** %b.addr, metadata !18, metadata !DIExpression()), !dbg !19
17     %0 = load i32*, i32** %b.addr, align 8, !dbg !20
18     %1 = load i32, i32* %0, align 4, !dbg !21
19     %2 = load i32*, i32** %a.addr, align 8, !dbg !22
20     store i32 %1, i32* %2, align 4, !dbg !23
21   ret void, !dbg !24
22 }
23
24 ; Function Attrs: nofree nosync nounwind readonly speculatable willreturn
25 declare void @_lvm.debug.declare(metadata, metadata, metadata) #1
26
27 ; Function Attrs: noinline nounwind optnone untable
28 define dso_local i32 @main() #0 !dbg !25 {
29   entry:
30   %x = alloca i32, align 4
31   %y = alloca i32, align 4
32   call void @_lvm.debug.declare(metadata i32* %x, metadata !28, metadata !DIExpression()), !dbg !29
33   store i32 2, i32* %x, align 4, !dbg !29
34   call void @_lvm.debug.declare(metadata i32* %y, metadata !30, metadata !DIExpression()), !dbg !31
35   store i32 3, i32* %y, align 4, !dbg !31
36   call void @ssign(i32* noundef %x, i32* noundef %y), !dbg !32
37   %0 = load i32, i32* %x, align 4, !dbg !33
38   %1 = load i32, i32* %y, align 4, !dbg !34
39   %call1 = call i32 (i8*, ...) @printf(i8* noundef getelementptr inbounds ([18 x i8], [18 x i8]* @.str, i64
40   ret i32 0, !dbg !36
41 }
```

Example of the LLVM IR output

# Context | WebSVF



Example of an Interprocedural Control Flow Graph (ICFG)

Enter your compile options:

-g x -c x -S x -fno-discard-value-names x -emit-llvm x

Select executable options:

saber (Memory Leak Detector) x

RUN

MEMORY LEAK: [1;33m NeverFree :[1;0m memory allocation at : ({ ln: 5 cl: 14 fl: example.c }) c

MEMORY LEAK: [1;33m NeverFree :[1;0m memory allocation at : (CallICFGNode: { "ln": 5, "cl": 14, "fl": "example.c" }) c

View Problem (Alt+F8) Quick Fix... (Ctrl+.)

```

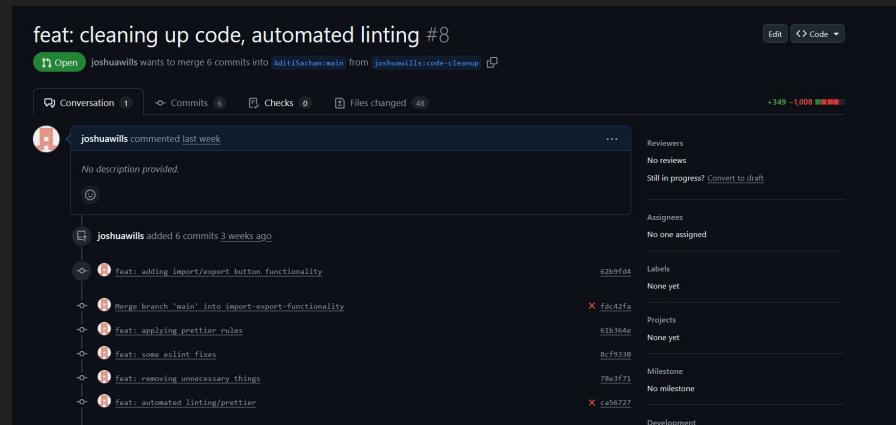
1 #include <std
2 #include <std
3
4 int main(void
5     int *x = malloc(sizeof(int) * 100);
6     x[0] = 100;
7 }
8

```

SABER executable identifying a memory leak

# Previous Features

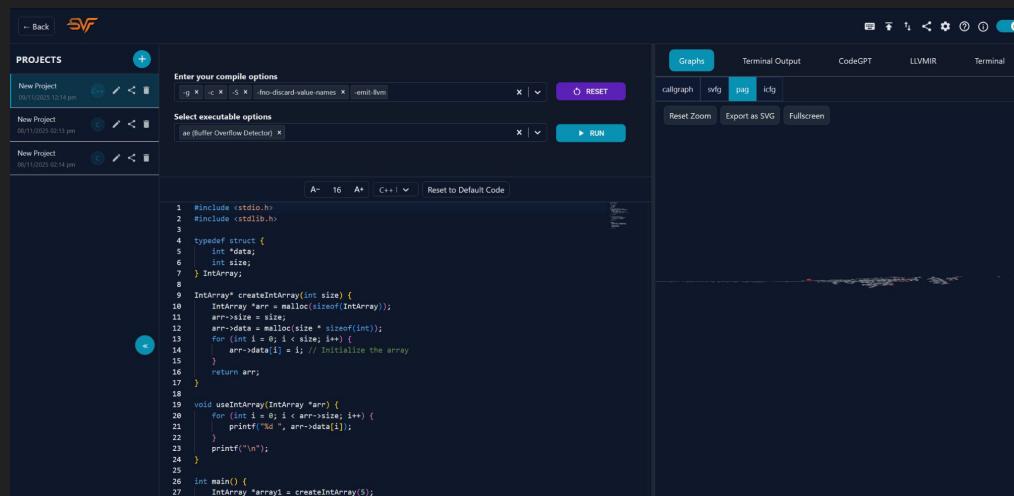
- Less Obvious Alterations
  - Software Clean Up
  - Python SVF Backend Conversion
  - Repository amalgamation
  - *All in the name of inviting contribution*



PR adding in cleaned up code

# Previous Features

- Import/Export Functionality
  - Interface with local filesystem
  - Streamline integration with existing workflow
- UI Enhancements
  - Modernise w/ consistency
  - Greater clarity in utility
  - Dark mode enhancements



The screenshot shows a dark-themed software interface, likely a developer's tool or IDE. At the top, there's a navigation bar with icons for Back, Home, and various settings. Below it is a 'PROJECTS' section listing three recent projects. The main area is a code editor with syntax highlighting for C/C++ code. The code is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct {
5     int *data;
6     int size;
7 } IntArray;
8
9 IntArray* createIntArray(int size) {
10     IntArray *arr = malloc(sizeof(IntArray));
11     arr->size = size;
12     arr->data = malloc(sizeof(int)*size);
13     for (int i = 0; i < size; i++) {
14         arr->data[i] = i; // Initialize the array
15     }
16
17     return arr;
18 }
19 void useIntArray(IntArray *arr) {
20     for (int i = 0; i < arr->size; i++) {
21         printf("%d ", arr->data[i]);
22     }
23     printf("\n");
24 }
25
26 int main() {
27     IntArray *array1 = createIntArray(5);
```

Below the code editor are two input fields: 'Enter your compile options' and 'Select executable options'. The 'Select executable options' field contains 'as (Buffer Overflow Detector)'. To the right of the code editor is a vertical toolbar with icons for Graphs, Terminal Output, CodeGPT, LLVMIR, and Terminal. Further down the page are buttons for 'Reset Zoom', 'Export as SVG', and 'Fullscreen'.

All the UI improvements in Thesis A and B

# **Josh's Improvements**

# C++ Support

**Why it was done:**

- Broaden scope of usability
- Increased functionality and support
- Easier onboarding for existing projects



# C++ Support

## What was done:

- Compilation support
  - Full breadth of LLVM IR, graphs, terminal output
- Languages specified for projects
- Ability to easily switch between them



# Custom Keybindings

## Why it was done:

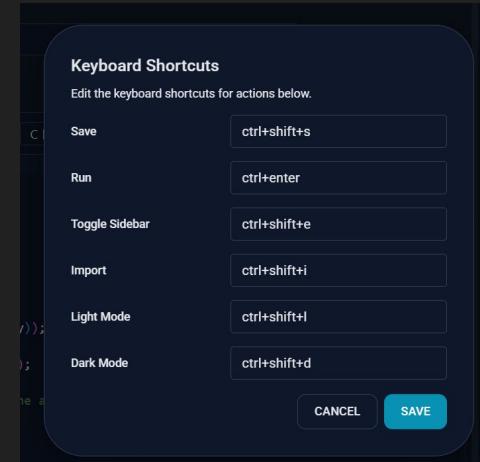
- Enhance the IDE-like feel of WebSVF
- Give users more freedom
- Enhance productivity and development speed



# Custom Keybindings

## What was done:

- Default keybindings for:
  - Save file, run SVF, toggle sidebar, import code, light mode, dark mode
- Ability to alter key bindings as one pleases

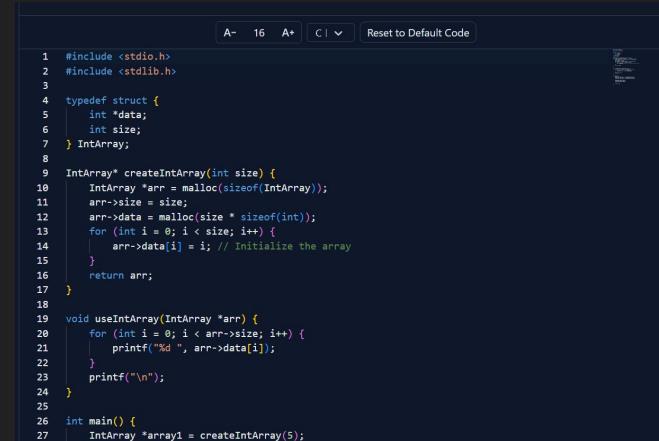


Modal to alter keyboard shortcuts on WebSVF

# Default Code Button

## Why it was done:

- Easy transition between C and C++ applications
- Ability to “revert” simply to an original state
- See an ideal example of a code snippet to test SVF with



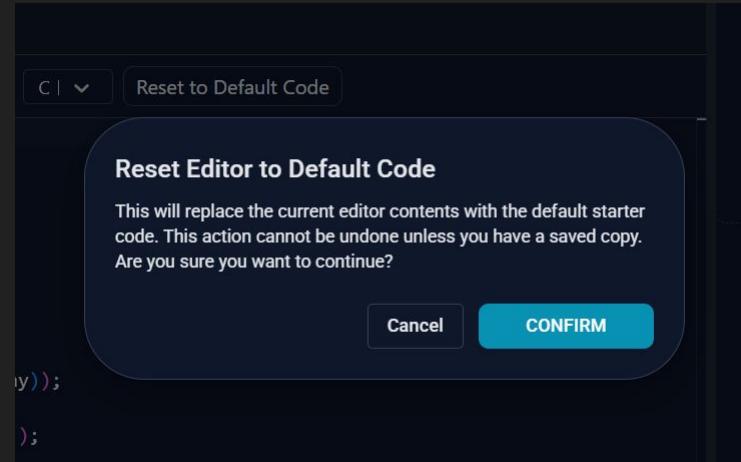
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct {
5     int *data;
6     int size;
7 } IntArray;
8
9 IntArray* createIntArray(int size) {
10    IntArray *arr = malloc(sizeof(IntArray));
11    arr->size = size;
12    arr->data = malloc(size * sizeof(int));
13    for (int i = 0; i < size; i++) {
14        arr->data[i] = i; // Initialize the array
15    }
16    return arr;
17 }
18
19 void useIntArray(IntArray *arr) {
20     for (int i = 0; i < arr->size; i++) {
21         printf("%d ", arr->data[i]);
22     }
23     printf("\n");
24 }
25
26 int main() {
27     IntArray *array1 = createIntArray(5);
```

Part of the default C code

# Default Code Button

## What was done:

- Button added to the text editor component to reset default code
- Confirmation modal to ensure code not accidentally lost
- Language specific defaults



Popup modal when you reset the code

# **Christians' Improvements**

# New User Tutorial/Onboarding Guide

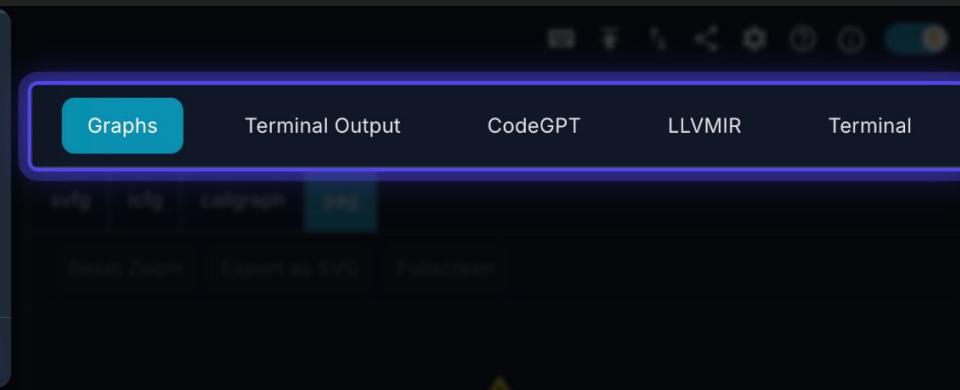
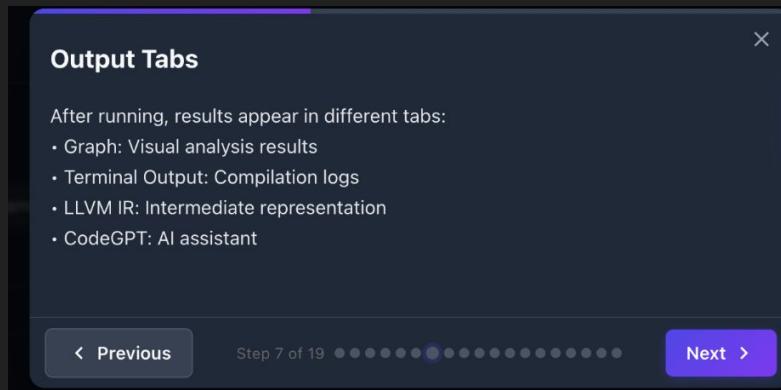
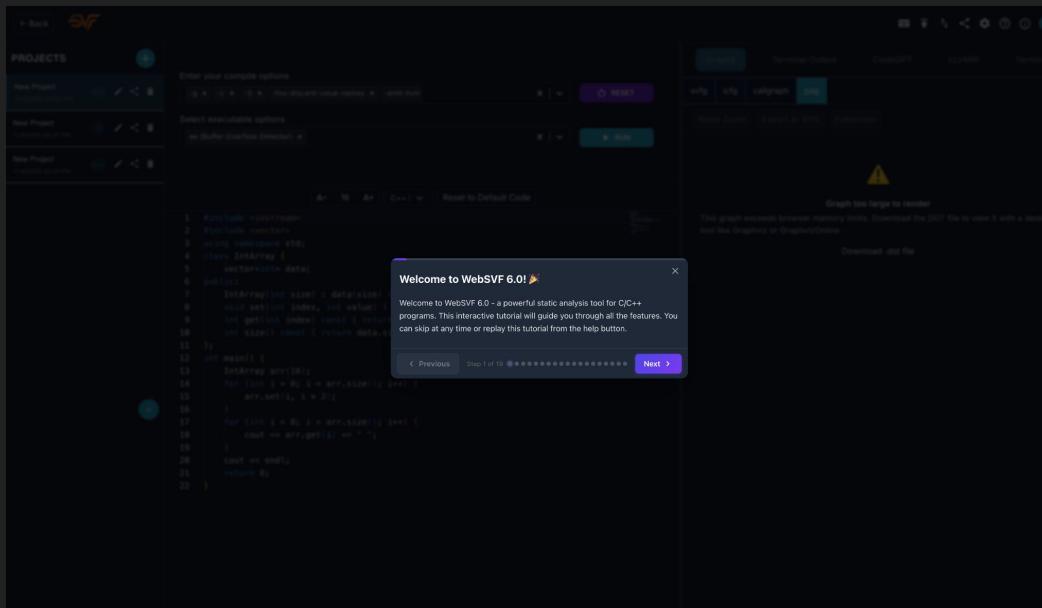
## Why it was done:

- Eases the transition for users to try out SVF's capabilities or integrate SVF into their software development lifecycle
  - Explains basic tools and features of SVF and WebSVF and how to use them
  - Simplifies key terminology essential to static analysis
- Adds professionalism to the WebSVF product, enticing users to use it

# New User Tutorial/Onboarding Guide

## What was done:

- When users first enter WebSVF, they are greeted with a tutorial, highlighting key features of WebSVF and allowing users to try it out themselves
- Tutorial can be replayed through navigation bar



# Terminal Feature Fix + Improvements

## Why it was done:

- Allows for greater use of Unix tools for extended analysis e.g. other analyses such as Valgrind
- Allows for greater integration with user's development cycle e.g. git for collaboration
- Gives a greater IDE feel, users can code and compile straight on the web

# Terminal Feature Fix + Improvements

## What was done:

- Integrated terminal per session that comes pre-installed with Unix tools e.g. git and Python
- Users are able to send their code written in WebSVF to this terminal for further command-line usage
- Ability to clear terminal and disconnect from it
- Included as part of tutorial

[Graphs](#)[Terminal Output](#)[CodeGPT](#)[LLVMIR](#)[Terminal](#)

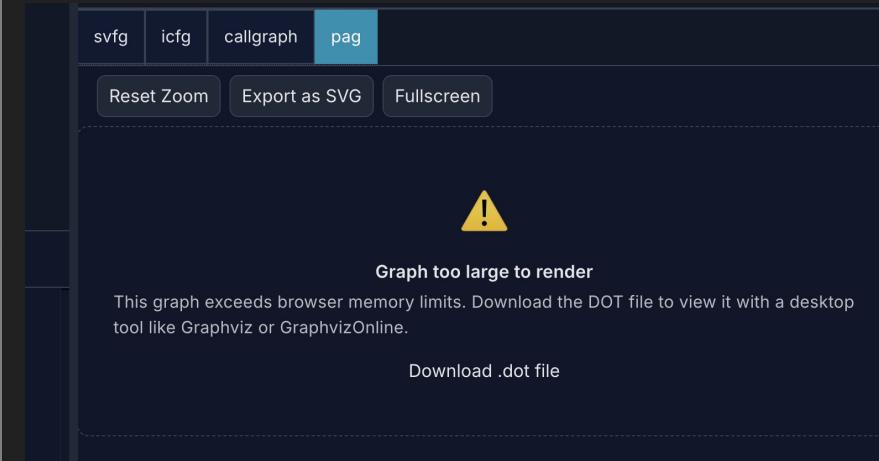
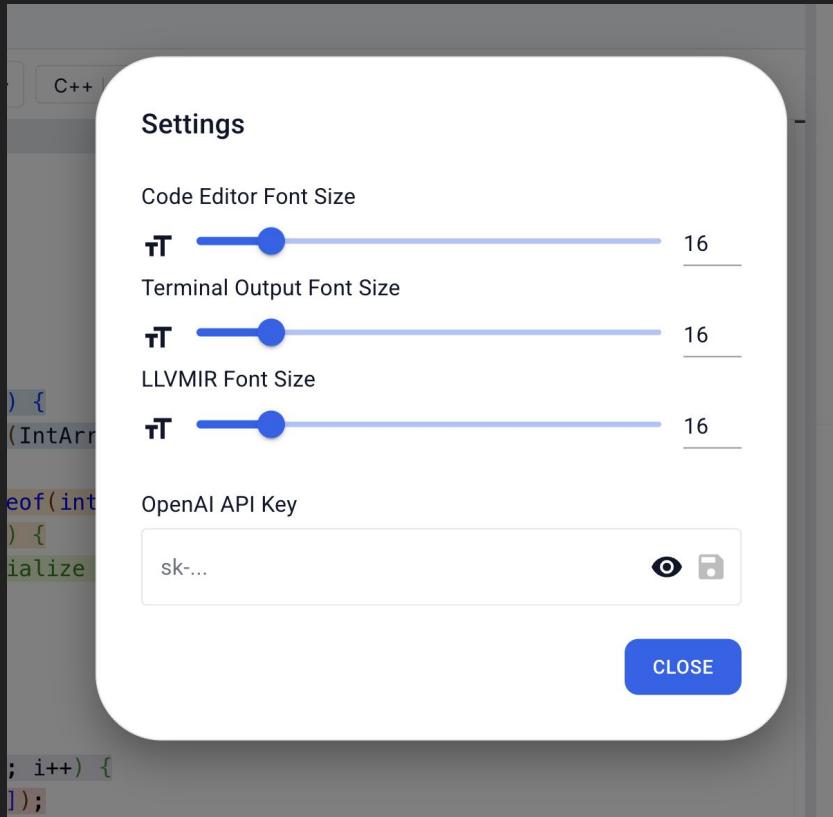
Terminal • Connected

[Clear](#)[Reconnect](#)[Paste Code](#)[Write code to terminal](#)

```
> #include <vector>
> using namespace std;
> class IntArray {
>     vector<int> data;
> public:
>     IntArray(int size) : data(size) {}
>     void set(int index, int value) { data[index] = value; }
>     int get(int index) const { return data[index]; }
>     int size() const { return data.size(); }
> };
> int main() {
>     IntArray arr(10);
>     for (int i = 0; i < arr.size(); i++) {
>         arr.set(i, i * 2);
>     }
>     for (int i = 0; i < arr.size(); i++) {
>         cout << arr.get(i) << " ";
>     }
>     cout << endl;
>     return 0;
> }
> EOF_7n5z0btw
root@48e2590f35e168:/app# ls
__pycache__  app.py  example.c  requirements.txt
root@48e2590f35e168:/app# █
```

# Miscellaneous Quality of Life Features

- Added new settings for users to enter their own OpenAI API key for extended usage
- Fixed multiple spacing, colouring and other styling issues
- Added labels to icon only buttons
- Error handling case when graphs generated exceed browser memory
- Bug fixes to sessions



# The Future of WebSVF

# Potential Improvements

- Ability to use different AI models e.g. Gemini, Grok, Claude
- Support more LLVM-IR compatible languages
- Support Multi-file projects and hybrid language projects
- GitHub/GitLab/Google Drive/OneDrive etc. integration
- Live collaboration e.g. Google Docs
- AI-assisted SVF command-line tool usage
- CI-like automated checks e.g. upload code to WebSVF, runs basic analysis as if it were a pipeline
- Code to Graph Node visual step-through mode
- Vim keybindings support in text-editor