

## 1. Clone subject applications

The complete list of subjects (along with their GitHub URLs) used for the evaluation of *LeakPair* is available in the [Evaluation Results](#) file. You may clone the subjects one by one and follow the subsequent steps.

## 2. Run the subject applications

### Installation and Build

After cloning a subject, you may want to follow the instructions on its GitHub repository that specify the commands to install the necessary dependencies and to get the app up and running. In case of both a SPA library or website, the build script is usually `build`. Please execute the exact build script as specified in the respective `package.json` file (and GitHub instructions) to verify the code correctness before and after *LeakPair* execution (RQ 3).

### Running a SPA website or web app

*LeakPair* repairs applications built by SPA frameworks (React and Angular). Hence, if the project contains both server-side and client-side repositories, you may only want to start the client-side code to replicate the results (though starting the server in parallel will have no effect). After installing the subject dependencies, the script to get the web app running is generally `start`, so you may want to use the command `yarn start` or `npm run start`. Please verify the actual script from the `package.json` file or GitHub instructions.

### Running a SPA library

SPA libraries are typically built and tested using [Storybook](#), a framework that allows developers to test working demos of components and libraries on a localhost port. If your subject is a SPA library, the script to execute it would most likely be `storybook`. Hence, the command to get the demo running would be `yarn storybook` or `npm run storybook`. Please verify the actual script from the `package.json` file or GitHub instructions.

## 3. Run the test suite (if available)

In order to answer RQ3 (*Do the patches by LeakPair break the functionality?*), in addition to a successful program build/compilation, we also need to track the test suite results before and after *LeakPair* changes (if such test scripts are provided in `package.json`). Hence, as the subject is running, open another terminal and execute the test script. In most applications, this script is simply `test`, so you may want to run the command `yarn test` or `npm run test`. To view the actual script, as well as verify if a test script is provided at all, please check the `package.json` file. Some projects also have end-to-end testing scripts (e2e). All such scripts should be run, and their results noted, in order to assess *LeakPair*'s impact on the program's functionality later on.

## 4. Run Memlab

Once the subject is up and running, you can now install and run Memlab on your system, following their [official documentation](#). Please note that Memlab is a dynamic analysis tool; hence, it requires that the application under test be in a running state (either live or locally).

After ensuring Memlab's installation, you can run Memlab on the subject app by providing it with the absolute path to the scenario file:

```
memlab run --scenario [path-to-the-scenario-file] -v
```

-v is optional but useful to see the details of the error in case Memlab terminates abruptly.

IMPORTANT: You may want to confirm that the initial (localhost) URL provided in the scenario file is the same as the one opened up in your browser after starting the app; otherwise, Memlab will throw an error.

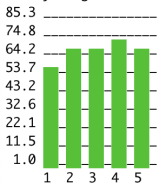
The complete list of scenario files for all the subject applications, is available in the [memlab-scenario-files](#) repository. Please select the respective scenario file by matching the name, for e.g, `roosterjs-scenario.js` for **roosterjs** project, `angular-components-scenario.js` for **angular-components** project, and so on.

### 5. Take note of memory footprints

On the completion of its execution, Memlab logs the detailed statistics of the memory footprint of the app, including the count of leaking objects, retainer clusters, and the final heap size. An example below:

```
page-load[59MB](baseline)[s1] > action-on-page[70.7MB] > revert[68.5MB] > action-on-page[74.2MB](target)[s4] > revert[68.7MB](final)[s5]
```

total time: 2min  
Memory usage across all steps:



detect and set JS snapshot engine: V8  
Alive objects allocated in target page:

(index)	name	type	count	retainedSize
0	'Detached HTMLDivElement'	'native'	11531	'6.5MB'
1	'Object'	'object'	14866	'2.2MB'
2	'Array'	'object'	15554	'2.2MB'
3	'StandaloneKeybindingService'	'object'	20	'1.4MB'
4	'KeybindingResolver'	'object'	20	'1.3MB'
5	'Detached HTMLSpanElement'	'native'	3686	'1.1MB'
6	'Detached FiberNode'	'object'	3935	'1MB'
7	''	'closure'	16444	'861KB'
8	'system / Map'	'object shape'	4197	'790.9KB'
9	'ResolvedKeybindingItem'	'object'	4380	'695.3KB'
10	'dispose'	'closure'	4681	'636KB'
11	'Map'	'object'	291	'573.4KB'
12	'Emitter'	'object'	2418	'415.2KB'
13	'StandaloneEditor'	'object'	20	'390.5KB'
14	'Detached Text'	'native'	2994	'299.4KB'
15	'Node'	'object'	4624	'296.1KB'
16	'Detached CSSStyleDeclaration'	'native'	4448	'278.6KB'
17	'Detached InternalNode'	'native'	4136	'252KB'
18	'TextModel'	'object'	20	'211KB'
19	'InternalEditorAction'	'object'	2280	'200.6KB'

- fiber node size: 1.4MB
- regular fiber node size: 0 byte
- detached fiber node size: 1.4MB
- alternate fiber node size: 0 byte
- error: 5.2KB

27477 Fiber nodes and Detached elements

27477 leaked objects

Sampling trace due to a large number of traces:

Number of Traces: 27477  
Sampling Ratio: 18.2%

MemLab found 9 leak(s)

As you will discover in the documentation, Memlab also outputs the logs in a file format along with heap snapshots and some other meta data. However, for the evaluation of *LeakPair*, taking note of the count of leaking objects/clusters and the final heap size in the CLI would suffice.

## 6. Run LeakPair

After noting the initial memory footprints of the subject, you may now stop the application. You would now run LeakPair on the subject. You should already be able to run LeakPair if you have Node version 14 or above installed in your system (refer to the [LeakPair - Installation and Usage](#) guide in the figshare dataset).

You can view the changes made by *LeakPair* using any diff tool if you want. An example of the Subscriptions leak fix in GitHub Desktop is shown below:

The screenshot displays the GitHub Desktop interface with a diff view for the file `src/app/core/common-components/template-tooltip/template-tooltip.directive.ts`. The left sidebar shows a list of 27 changed files, including `src/app/core/common-components/template-tooltip.directive.ts` which is selected. The main area shows the diff between two versions of the file. The changes include:

- Imports: `import { ComponentPortal } from "@angular/cdk/portal";` and `import { TemplateTooltipComponent } from "../template-tooltip.component";`.
- Imports: `import { takeUntil } from "rxjs/operators";` and `import { Subject } from "rxjs";`.
- Comments: `/** A directive that can be used to render a custom tooltip that may contain HTML code. * When a tooltip is only a string, the {code MatTooltip} should be used instead. */`.
- Class Definition: `export class TemplateTooltipDirective implements OnInit, OnDestroy {`
- Private Property: `private _destroy$: Subject<any> = new Subject<any>();`
- Methods: `ngOnDestroy() {` containing `this._destroy$.next(true);`, `this._destroy$.complete();`, and `this.hide();`.
- Class Closure: `}`

At the bottom, a summary section indicates a write access error for 'ndb-core' and suggests creating a fork.

## 7. Re-run the subject applications (+ test suite)

Once *LeakPair* has made its changes, you need to restart the application by following the same commands as before.

### RQ3: Do the patches by *LeakPair* break the functionality?

The subject should build and/or compile successfully after the changes. Also, if there were any test scripts available, run them again now (in another terminal, as done before). The count of passed/failing tests should remain unchanged.

## 8. Re-run Memlab

While the subject is still running, you can now run Memlab again with the same command and scenario file.

## **9. Take note of memory footprints (after *LeakPair* changes)**

### **RQ1: How effective is *LeakPair*?**

Take note of the count of leaking objects and the final heap size this time. As *LeakPair* bases its efficacy on the reduction in the count of leaking objects or the heap size, at least one of them should show a noticeable reduction (in our experiments, the count of leaking objects frequently showed a more significant reduction).

### **RQ2: Are the patches by *LeakPair* acceptable?**

As the review process is double-blinded, we plan to disclose the submitted PRs after the final decision to preserve anonymity, since the PRs were all created by the author's personal GitHub account (the status of the PRs at the time of submission, as well as the answer to this RQ are presented in Section 5.2 in the paper).