NC STATE

EasyView: Bridging Software Developers with Dynamic Program Analysis

Xu Liu
xliu88@ncsu.edu
Associate Professor, CS department
NC State University
ORNL JFA

Diverse Performance Tools

For HPC

HPCToolkit, TAU, Caliper, Score-P, …

For cloud

Cloud Profiler, DynaTrace, …

For system

Perf, VTune, OProfile, uProf, NSight Compute, rocProfiler, …

For fine-grained analysis

Pin, Valgrind, NVBit, GTPin, DynamoRIO, …

For different languages

Pyinstrument, Scalene, cProfile for Python

Async-Profiler, JXPerf for Java

PProf for Go

But, There is a Gap

Tool developer's perspective

- Focus on different capabilities
- Require efforts to implement redundant features: e.g., GUI

Tool user's perspective

- Have too many choices
- Need expert knowledge, deep learning curve
- Customize the analysis upon the diverse demands
- Switch between tools and development environments

Efforts on Bridging the Gap

Unifying data collection

- Linux timer
- Linux Perf_events

Standardizing tools interface

- PMPI, OMPT in HPC domain

But, one most important part missing: unified data analysis and visualization framework

Our Approach: EasyView

EasyView bridges the gaps for analysis and visualization

EasyView features

- Unify various profilers in a single framework
- Integrate program analysis with development environment (IDE/browser)
- Support various analyses

Benefits of EasyView

- Facilitate performance data interpretation
  - Unified GUI, IDE integration
- Analyze a program measured by different tools
  - Powerful analysis
- Reduce tool development efforts on data analysis and visualization

EasyView Design Principles

Principles

Easy: easy to install and use

General: generalize data format with profiling IR

Insightful: support insightful analysis with various views

Efficient: provide smooth user experiences in exploring the data

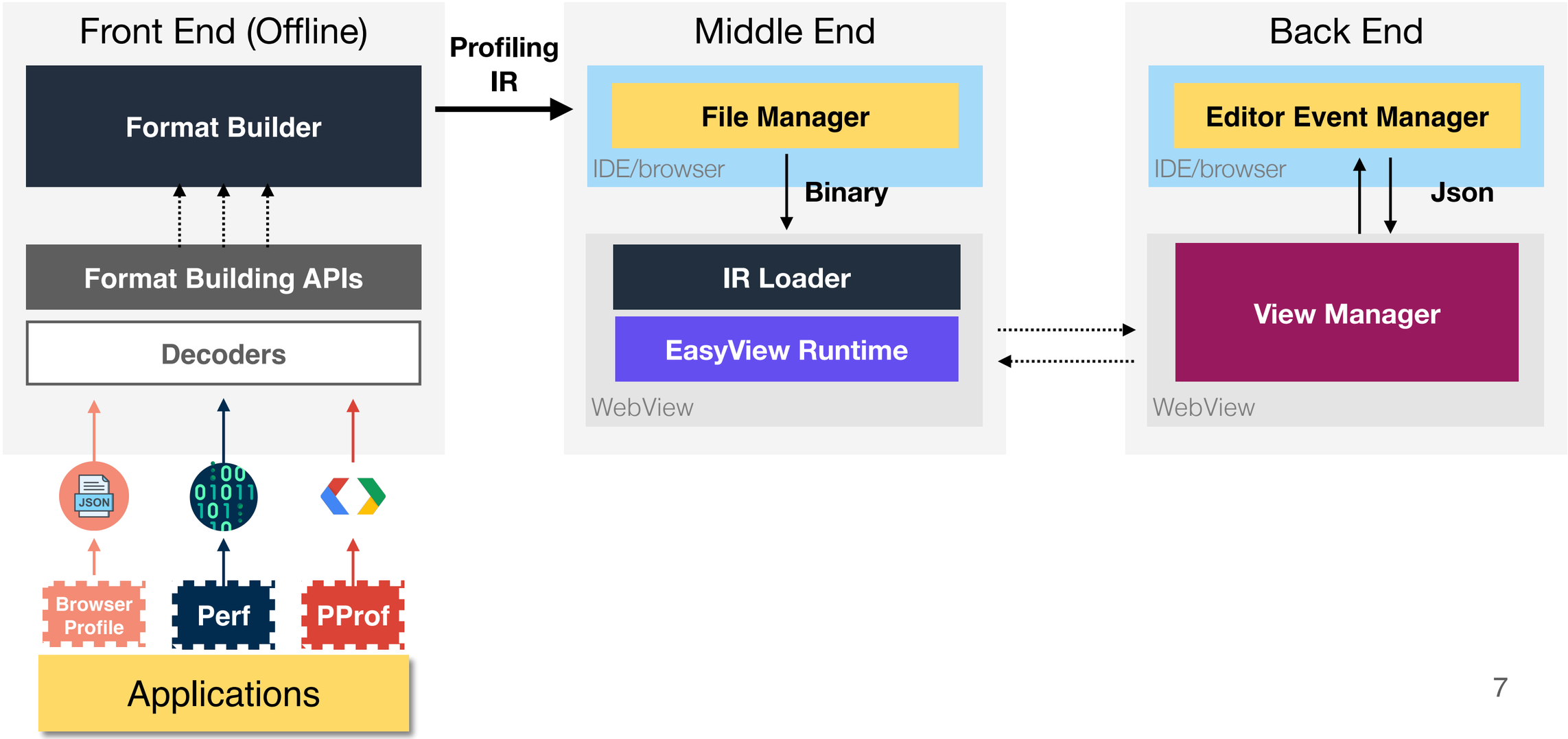
Applicable: integrate into IDE or browser with web techniques

Extensible: support customized analysis

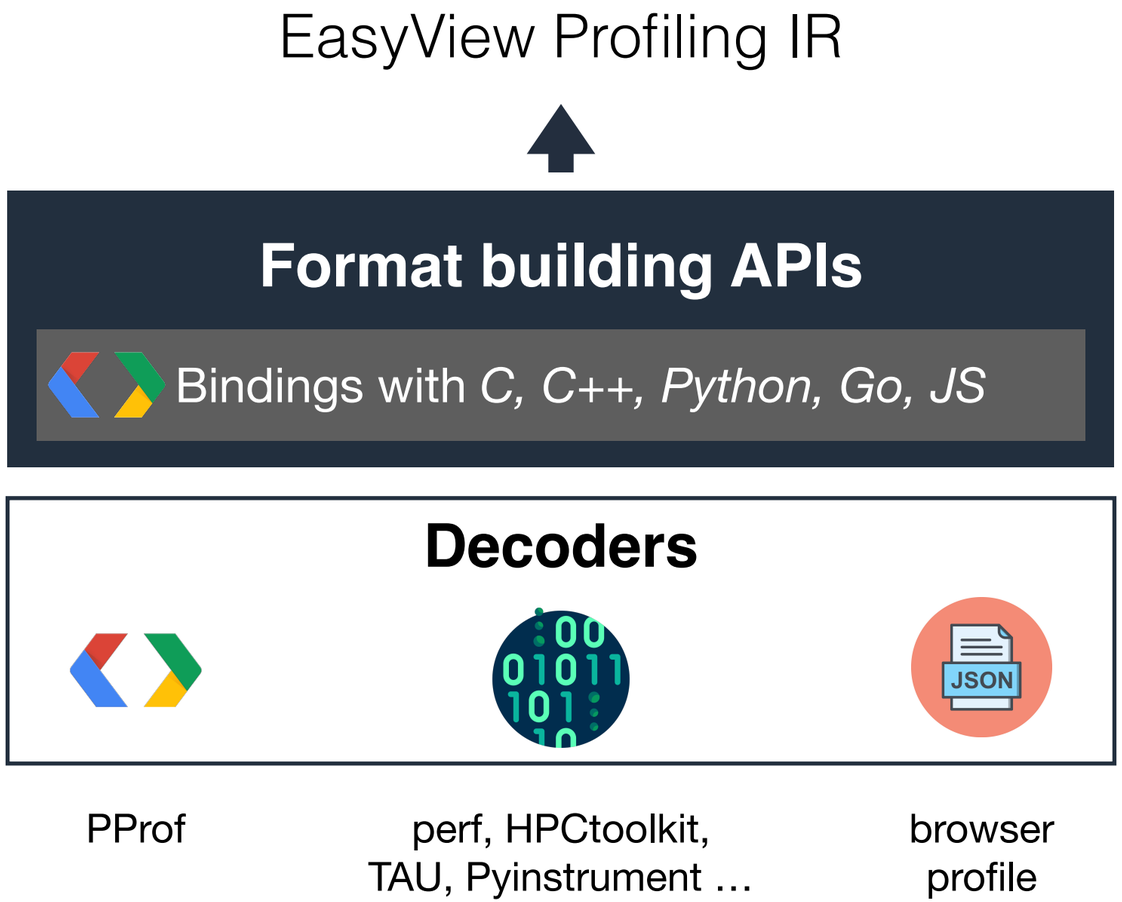
Intelligent: integrate ML models to aid analysis

Secure: process profiles locally

EasyView Overview

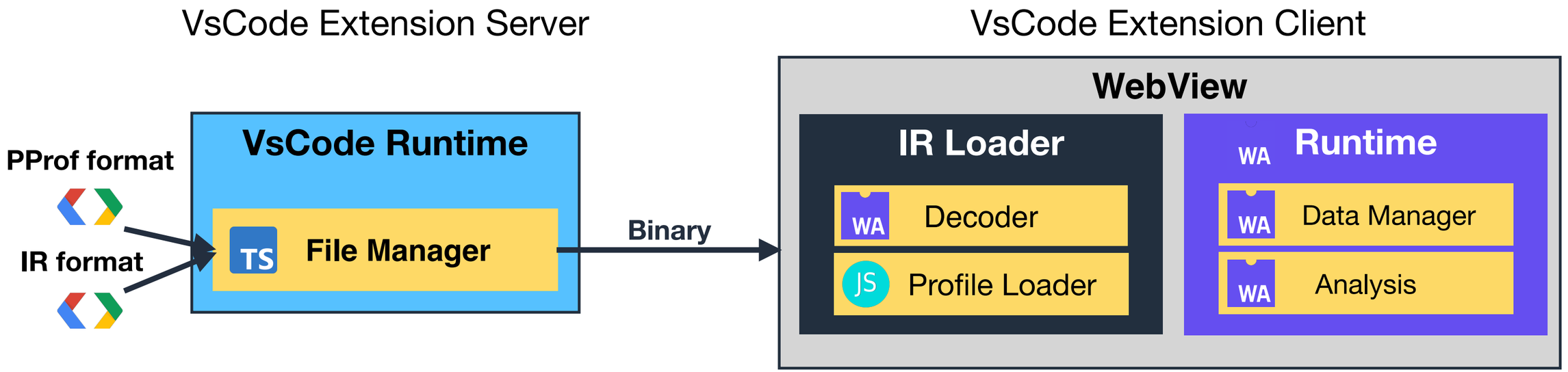


EasyView Front End



https://github.com/Xuhpclab/drcctprof-databuilder

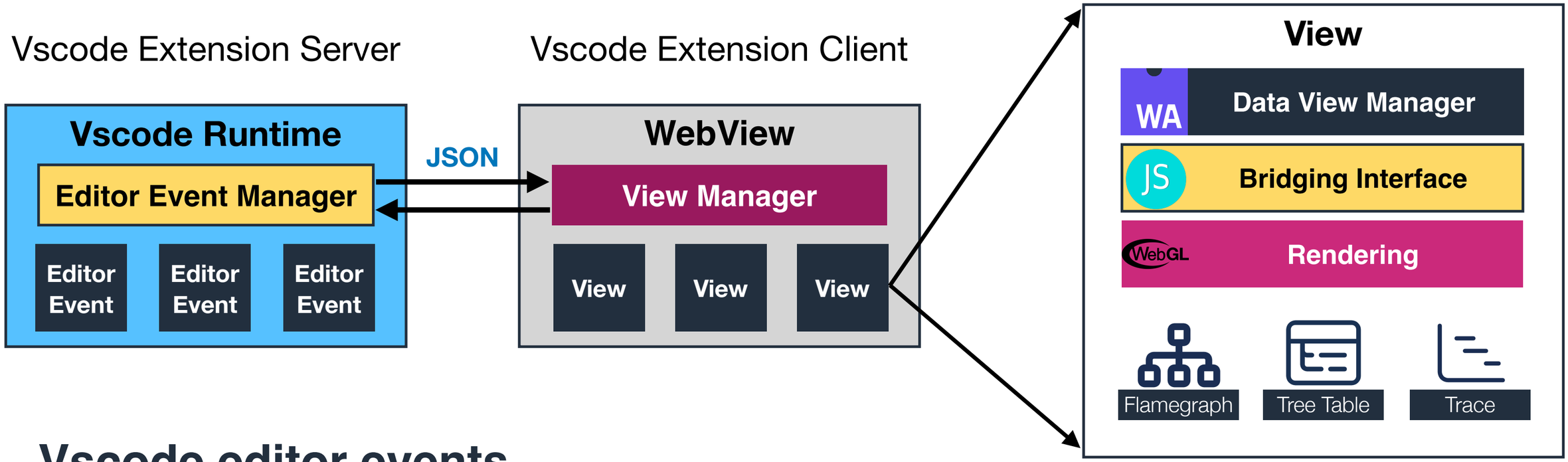
EasyView Middle End Design



Optimization for efficiency

- Abandon JS (or TS) for computation-intensive processing
  - Use web assembly instead
- Avoid passing data in json format
- Apply many other optimizations for memory and computation efficiency

EasyView Back End



Vscode editor events

- UI operations: highlights, codelens
- File operations: code link, file open/close/switch
- Other operations: hovers, popups

Performance Evaluation

EasyView vs. PProf

- Profile collected from a real industrial workload
- Machine configuration
  - CPU: 3.8 GHz 8-Core Intel Core i7
  - Memory: 64 GB 2667 MHz DDR4
  - GPU: AMD Radeon Pro 5500 XT 8 GB

EasyView is far more efficient than PProf

Demo

Conclusions

EasyView, a powerful analysis and visualization tool

- Bridge tool developers and users
- Integrate into IDEs with mostly web front end techniques
- Can be extended with various analysis
- General, insightful, efficient, applicable

Side products of EasyView

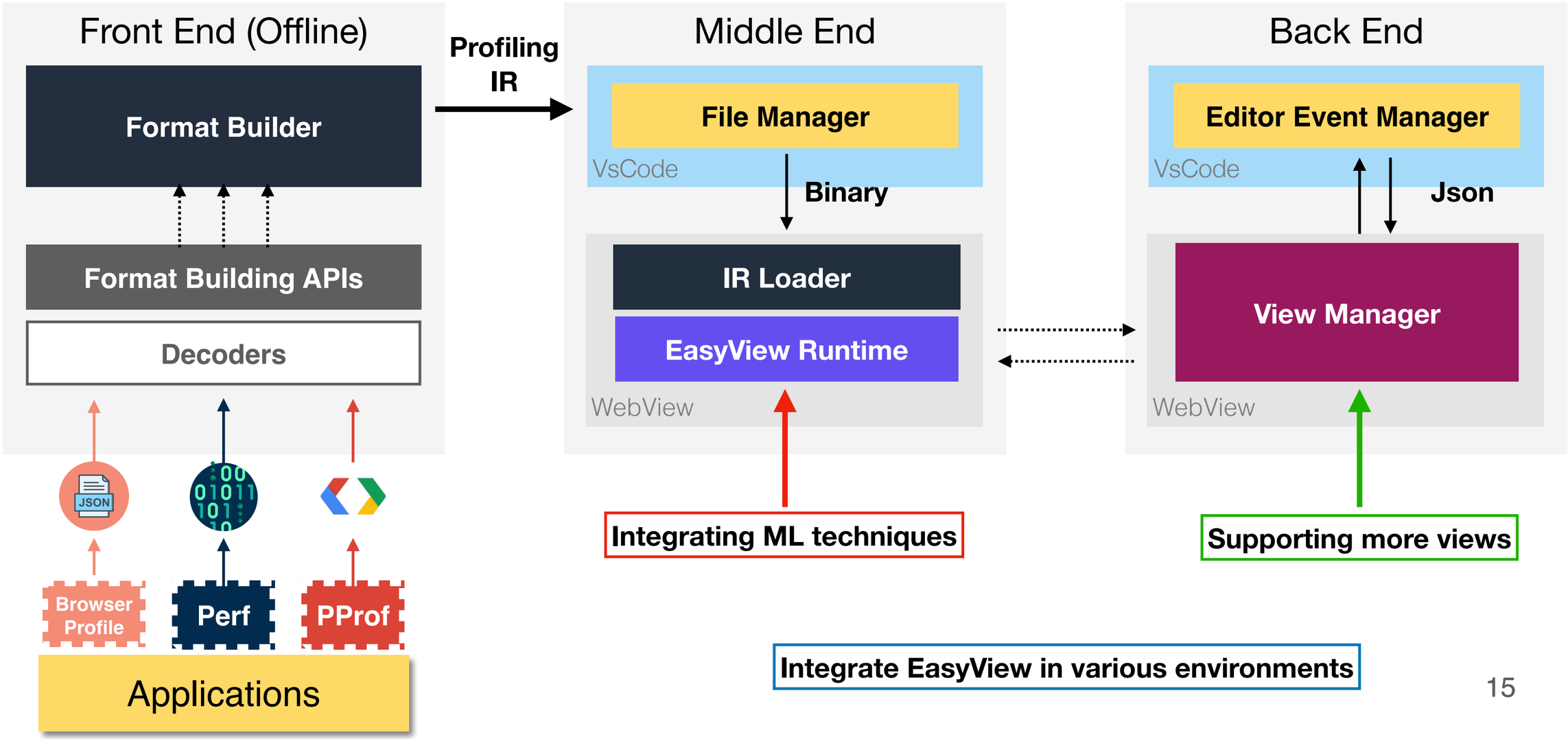
- Profiling format: compact and informative
- Flame graph visualization component: fast

Research contributions

- Prove that pure web front end techniques provide sufficient computation power to analyze and visualize massive amounts of performance data

https://www.easyview.dev

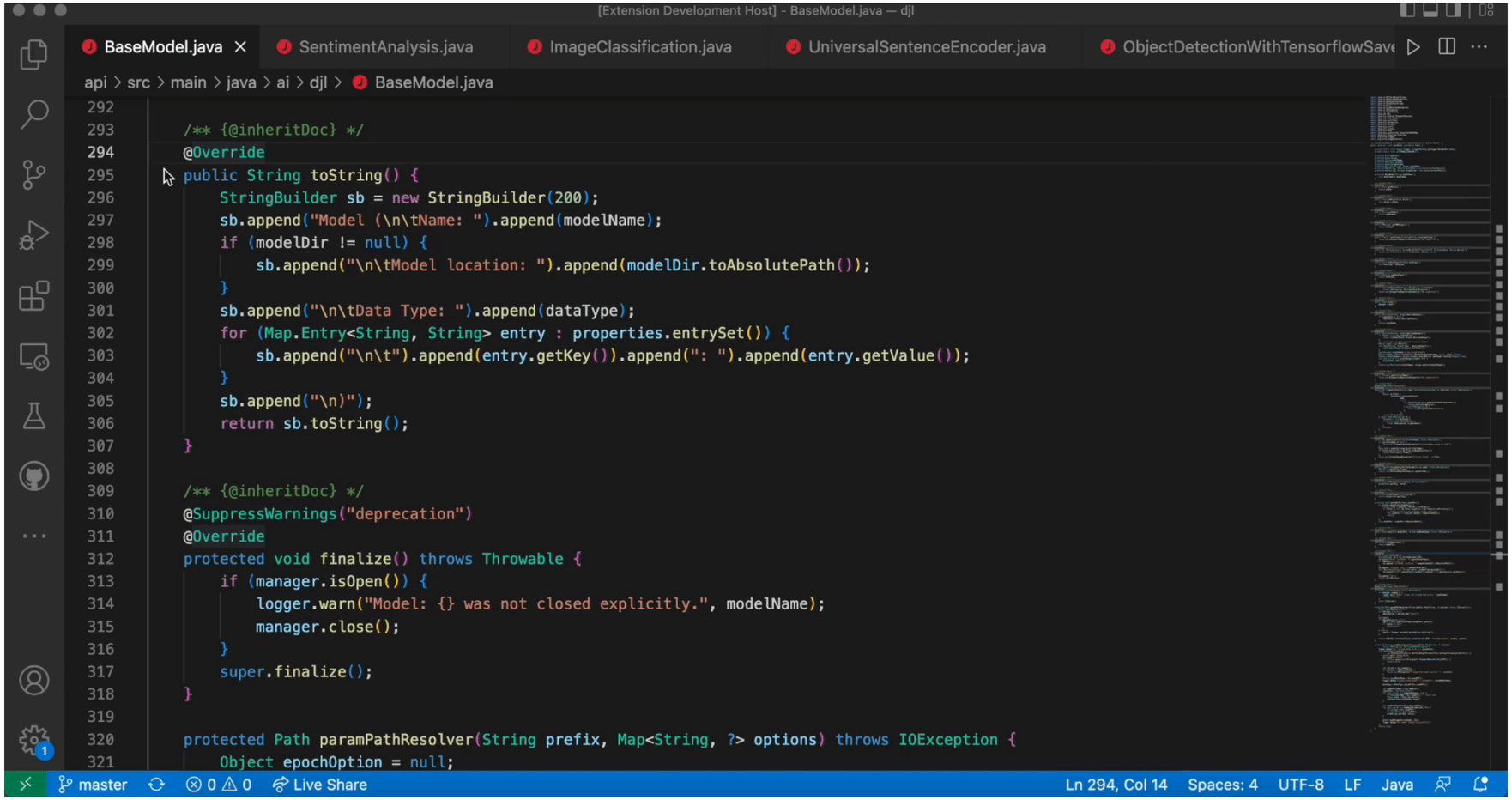
On-Going Work



AI-Aided Analysis

Code Summary for Single Function

Select a whole function




```java
    /** {@inheritDoc} */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder(200);
        sb.append("Model (\n\tName: ").append(modelName);
        if (modelDir != null) {
            sb.append("\n\tModel location: ").append(modelDir.toAbsolutePath());
        }
        sb.append("\n\tData Type: ").append(dataType);
        for (Map.Entry<String, String> entry : properties.entrySet()) {
            sb.append("\n\t").append(entry.getKey()).append(": ").append(entry.getValue());
        }
        sb.append("\n)");
        return sb.toString();
    }

    /** {@inheritDoc} */
    @SuppressWarnings("deprecation")
    @Override
    protected void finalize() throws Throwable {
        if (manager.isOpen()) {
            logger.warn("Model: {} was not closed explicitly.", modelName);
            manager.close();
        }
        super.finalize();
    }

    protected Path paramPathResolver(String prefix, Map<String, ?> options) throws IOException {
        Object epochOption = null;
```