Supplementary Materials

A Practical Human Labeling Method for Online Just-in-Time Software Defect Prediction

LIYAN SONG, Southern University of Science and Technology, Shenzhen, China
LEANDRO L. MINKU*, The University of Birmingham, Edgbaston, Birmingham, UK
CONG TENG, Southern University of Science and Technology, Shenzhen, China
XIN YAO*, Southern University of Science and Technology, Shenzhen, China

This supplementary material complements the paper entitled "A Practical Human Labeling Method for Online Just-in-Time Software Defect Prediction" published in ESEC/FSE'23.

We report overall information of the 14 GitHub open source projects (datasets) in Section 1 of this supplementary material, including the summary Table 1, the traditional features extracted by `Commit Guru`, and the preprocessing on these features. We also reports more comprehensive result tables and plots relating to RQ1.1, RQ1.2, RQ1.3, and RQ2.1 in Section 2 of this supplementary material, including the predictive performance of JIT-SDP in terms of various evaluation metrics. The full experimental results of RQ2.2 and RQ2.3 are available in the main paper; therefore, there is no need to provide additional details in this supplementary material.

*Liyan Song, Cong Teng, and Xin Yao (Corresponding Author) are with Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, Shenzhen, China and Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China.

Leandro L. Minku (Corresponding Author) is with School of Computer Science, the University of Birmingham, Edgbaston, Birmingham, UK

Manuscript submitted to ACM

1 DATASETS

This paper uses 14 GitHub open source projects similar to previous work [4, 5] to investigate the proposed human labeling methods for JIT-SDP, as summarized in Table 1. They were chosen randomly among GitHub projects with more than 4 years of duration, rich history (>10k commits) and a wide range of defect-inducing change ratio (2%~45%). The first 7 projects were made available in [1] and the rest 7 projects were made available in [4].

A previous study [5] showed that, if we use the first 10k changes of the projects in our study, there is at least an estimated 99% confidence level that the fixes corresponding to these changes have already been reported. Therefore, we use the first 10k software changes of each project in the experiments to increase data quality. The second and third columns of Table 1 list the total number of collected software changes and the percentage of defect-inducing software changes over the total number of changes for each project, respectively. The fourth column of Table 1 lists the percentage of defect-inducing software changes over the first 10k changes for each project.

Table 1. An overview of the datasets. The first 10,000 (10k) software changes are used in our experiments.

Dataset	Total Changes	Defect% (all)	Defect% (first 10k)	Time Period	Main Language
Brackets	11,601	34.02	36.39	12/2011 - 12/2017	JavaScript
Broadleaf	12,336	20.28	22.93	11/2008 - 12/2017	Java
Camel	30,229	20.67	34.5	03/2007 - 12/2017	Java
Fabric8	12,495	20.65	21.11	04/2011 - 12/2017	Java
jGroups	18,003	20.34	11.3	09/2003 - 12/2017	Java
Nova	26,312	44.34	52.95	05/2010 - 01/2018	Python
Tomcat	18,721	27.81	33.25	03/2006 - 12/2017	Java
Corefx	26,627	6.91	6.85	11/2014 - 11/2019	C#
Django	26,352	42.65	47.89	07/2005 - 09/2019	Python
Rails	57,944	25.64	36.85	11/2004 - 09/2019	Ruby
Rust	73,876	2.02	6.30	06/2010 - 10/2019	Rust
Tensorflow	65,034	24.85	30.26	11/2015 - 01/2020	C++
VScode	51,846	2.28	3.82	11/2015 - 10/2019	TypeScript
wp-Calypso	31,206	22.75	24.96	05/2014 - 10/2019	JavaScript

The software change features consist of 14 metrics that can be grouped into 5 dimensions as (1) diffusion: NS (number of modified subsystems), ND (number of modified directories), NF (number of modified files) and Entropy (distribution of modified code across each file), (2) size: LA (lines of code added), LD (lines of code deleted) and LT (lines of code in a file before the change), (3) purpose: FIX (whether or not the change is to fix a defect), (4) history: NDEV (number of developers that changed the modified files), AGE (average time interval between the last and the current change) and NUC (number of unique changes to the modified files) and (5) experience: EXP (developer experience), REXP (recent developer experience) and SEXP (developer experience on a subsystem).

These metrics have shown to be good indicators for JIT-SDP [2]. Prior studies have also recommended to preprocess the 14 feature metrics for better predictive performance in JIT-SDP [2, 7]. We produce 12 transformed feature metrics following the same preprocessing procedures as Kamei et al. in [2] as

(1) *Removing highly correlated features*: (i) LA and LD are normalized by dividing by LT as also recommended by Nagappan and Ball [3]. (ii) LT and NUC are normalized by dividing by NF since LT and NUC are highly correlated with NF. (iii) ND and REXP are removed as they are highly correlated with NF and EXP, respectively.
(2) *Dealing with skew*: Since most features are highly skewed, each metric went through logarithmic transformation except for FIX (a Boolean variable).

2 EXPERIMENTAL RESULTS

2.1 RQ1: JIT-SDP with HumLa

2.1.1 Additional Result Tables. Tables in this section report additional performance results for RQ1 in various evaluation metrics including G-Mean, MCC, recall 0, recall 1, precision and F1 score, providing further insights for readers of interest in a more detailed analysis of the results.

Table 2. RQ1.1 & RQ1.2 – Average G-Mean of JIT-SDP with HumLa at different amounts of human noise across 100 runs. The last row reports the statistical tests across datasets. JIT-SDP with HumLa at 0-human noise is chosen as the control method. Significant difference against the control method is highlighted in yellow (light gray). Smaller rankings represent better predictive performance for JIT-SDP when there is statistically significant difference.

Dataset	Waiting time	HumLa at different amounts of human noise										
		0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Bracket	0.643	0.639	0.641	0.642	0.640	0.640	0.638	0.641	0.639	0.637	0.633	0.627
Broadleaf	0.607	0.663	0.661	0.654	0.647	0.637	0.626	0.614	0.604	0.593	0.582	0.570
Camel	0.669	0.681	0.678	0.676	0.670	0.667	0.669	0.667	0.664	0.660	0.653	0.649
Fabric8	0.653	0.661	0.659	0.658	0.657	0.655	0.654	0.652	0.647	0.641	0.633	0.623
jGroup	0.568	0.600	0.596	0.591	0.586	0.582	0.578	0.574	0.570	0.563	0.552	0.537
Nova	0.682	0.688	0.689	0.689	0.689	0.688	0.687	0.684	0.684	0.678	0.668	0.650
Tomcat	0.613	0.638	0.637	0.637	0.636	0.633	0.630	0.628	0.623	0.615	0.596	0.573
Corefx	0.639	0.636	0.638	0.634	0.632	0.633	0.626	0.625	0.622	0.618	0.614	0.607
Django	0.690	0.698	0.697	0.696	0.696	0.697	0.698	0.698	0.696	0.691	0.680	0.665
Rails	0.562	0.623	0.626	0.625	0.623	0.616	0.605	0.594	0.583	0.569	0.554	0.539
Rust	0.584	0.586	0.589	0.589	0.590	0.586	0.587	0.584	0.579	0.573	0.565	0.564
Tensorflow	0.691	0.678	0.676	0.675	0.677	0.683	0.688	0.691	0.690	0.684	0.674	0.663
VScode	0.527	0.527	0.524	0.520	0.515	0.514	0.509	0.500	0.493	0.486	0.479	0.469
wp-Calypso	0.551	0.622	0.622	0.618	0.615	0.608	0.598	0.583	0.565	0.538	0.496	0.453
aveRank	6.39 (*)	2.79	2.89	3.61	4.29	5.32	5.75	6.46	8.07	9.43	11.00	12.00

Table 3. RQ1.1 & RQ1.2 – Average MCC of JIT-SDP with HumLa at different amounts of human noise across 100 runs. Please refer to Table 2 for more description.

Dataset	Waiting time	HumLa at different amounts of human noise										
		0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Bracket	0.300	0.297	0.302	0.302	0.300	0.299	0.299	0.305	0.302	0.299	0.292	0.282
Broadleaf	0.292	0.347	0.343	0.336	0.328	0.320	0.310	0.303	0.294	0.286	0.279	0.270
Camel	0.356	0.378	0.375	0.371	0.366	0.359	0.359	0.353	0.348	0.341	0.329	0.321
Fabric8	0.320	0.333	0.331	0.329	0.328	0.325	0.322	0.320	0.312	0.301	0.289	0.275
jGroup	0.193	0.242	0.238	0.232	0.227	0.223	0.221	0.219	0.218	0.215	0.208	0.202
Nova	0.380	0.391	0.392	0.393	0.392	0.389	0.387	0.383	0.382	0.375	0.363	0.345
Tomcat	0.282	0.309	0.309	0.309	0.308	0.304	0.301	0.298	0.293	0.285	0.269	0.247
Corefx	0.362	0.360	0.357	0.353	0.351	0.350	0.340	0.345	0.344	0.338	0.344	0.344
Django	0.413	0.430	0.427	0.427	0.427	0.429	0.432	0.430	0.429	0.422	0.408	0.389
Rails	0.214	0.296	0.286	0.278	0.279	0.268	0.253	0.244	0.235	0.226	0.217	0.205
Rust	0.250	0.248	0.252	0.256	0.257	0.258	0.259	0.256	0.256	0.249	0.248	0.249
Tensorflow	0.389	0.394	0.390	0.388	0.390	0.394	0.395	0.393	0.388	0.378	0.367	0.355
VScode	0.276	0.302	0.297	0.291	0.287	0.283	0.279	0.269	0.263	0.254	0.251	0.242
wp-Calypso	0.256	0.293	0.293	0.290	0.290	0.286	0.283	0.274	0.267	0.254	0.235	0.215
aveRank	8.04 (*)	3.11	2.96	3.82	4.29	4.82	5.46	6.14	7.36	9.64	10.86	11.50

Table 4. RQ1.1 & RQ1.2 – Average Recall 1 of JIT-SDP with HumLa at different amounts of human noise across 100 runs. Please refer to Table 2 for more description.

Dataset	Waiting time	HumLa at different amounts of human noise										
		0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Bracket	0.652	0.638	0.647	0.650	0.655	0.652	0.647	0.625	0.602	0.589	0.576	0.556
Broadleaf	0.470	0.605	0.595	0.573	0.552	0.528	0.502	0.478	0.459	0.439	0.420	0.401
Camel	0.700	0.736	0.736	0.728	0.733	0.728	0.722	0.711	0.706	0.696	0.676	0.661
Fabric8	0.639	0.662	0.665	0.664	0.660	0.655	0.647	0.637	0.620	0.601	0.574	0.548
jGroup	0.471	0.517	0.507	0.494	0.482	0.475	0.465	0.456	0.443	0.422	0.399	0.365
Nova	0.646	0.660	0.659	0.656	0.655	0.656	0.653	0.643	0.640	0.618	0.588	0.547
Tomcat	0.508	0.629	0.623	0.614	0.608	0.594	0.582	0.567	0.542	0.507	0.459	0.412
Corefx	0.485	0.480	0.486	0.480	0.476	0.478	0.468	0.464	0.456	0.451	0.438	0.426
Django	0.600	0.620	0.616	0.613	0.613	0.615	0.612	0.609	0.600	0.587	0.563	0.535
Rails	0.465	0.764	0.730	0.683	0.650	0.599	0.557	0.518	0.484	0.448	0.415	0.386
Rust	0.452	0.457	0.462	0.457	0.460	0.448	0.446	0.443	0.427	0.421	0.401	0.397
Tensorflow	0.683	0.800	0.799	0.796	0.791	0.777	0.756	0.725	0.694	0.657	0.615	0.577
VScode	0.347	0.328	0.323	0.319	0.311	0.310	0.302	0.292	0.282	0.273	0.264	0.252
wp-Calypso	0.363	0.513	0.518	0.513	0.505	0.479	0.452	0.419	0.384	0.341	0.281	0.231
aveRank	6.64 (*)	2.21	2.00	3.14	3.93	4.71	6.29	7.50	8.57	10.00	11.00	12.00

Table 5. RQ1.1 & RQ1.2 – Average recall 0 of JIT-SDP with HumLa at different amounts of human noise across 100 runs. Please refer to Table 2 for more description.

Dataset	Waiting time	HumLa at different amounts of human noise										
		0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Bracket	0.645	0.654	0.650	0.647	0.640	0.642	0.646	0.673	0.693	0.703	0.709	0.718
Broadleaf	0.802	0.736	0.741	0.754	0.766	0.779	0.791	0.805	0.813	0.823	0.831	0.839
Camel	0.650	0.636	0.633	0.636	0.625	0.623	0.630	0.636	0.635	0.639	0.647	0.652
Fabric8	0.678	0.669	0.662	0.661	0.664	0.666	0.671	0.679	0.687	0.696	0.709	0.719
jGroup	0.710	0.716	0.720	0.727	0.732	0.735	0.742	0.747	0.758	0.773	0.786	0.811
Nova	0.729	0.727	0.728	0.732	0.732	0.728	0.729	0.735	0.738	0.751	0.767	0.784
Tomcat	0.762	0.671	0.677	0.686	0.690	0.701	0.710	0.722	0.741	0.765	0.793	0.812
Corefx	0.849	0.853	0.845	0.846	0.847	0.844	0.844	0.851	0.856	0.855	0.869	0.878
Django	0.803	0.799	0.800	0.802	0.803	0.803	0.809	0.810	0.816	0.822	0.830	0.835
Rails	0.730	0.519	0.548	0.588	0.621	0.661	0.687	0.714	0.737	0.759	0.780	0.794
Rust	0.778	0.772	0.772	0.779	0.778	0.788	0.791	0.791	0.804	0.803	0.818	0.822
Tensorflow	0.703	0.582	0.579	0.580	0.588	0.609	0.634	0.664	0.691	0.718	0.746	0.769
VScode	0.869	0.906	0.907	0.906	0.909	0.908	0.911	0.911	0.915	0.916	0.920	0.923
wp-Calypso	0.858	0.769	0.763	0.764	0.771	0.790	0.811	0.830	0.851	0.873	0.903	0.925
aveRank	6.86 (*)	9.93	10.43	9.36	9.00	8.86	7.43	5.50	4.36	3.21	2.07	1.00

Table 6. RQ1.1 & RQ1.2 – Average Precision of JIT-SDP with HumLa at different amounts of human noise across 100 runs. Please refer to Table 2 for more description.

Dataset	Waiting time	HumLa at different amounts of human noise										
		0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Bracket	0.653	0.654	0.654	0.653	0.651	0.651	0.654	0.664	0.669	0.671	0.670	0.669
Broadleaf	0.710	0.701	0.702	0.704	0.706	0.710	0.711	0.715	0.716	0.718	0.720	0.720
Camel	0.672	0.673	0.671	0.671	0.667	0.665	0.666	0.667	0.665	0.665	0.664	0.663
Fabric8	0.666	0.668	0.665	0.664	0.665	0.664	0.665	0.667	0.667	0.667	0.667	0.666
jGroup	0.634	0.654	0.653	0.654	0.653	0.654	0.656	0.658	0.661	0.665	0.667	0.673
Nova	0.710	0.713	0.715	0.716	0.716	0.713	0.712	0.713	0.714	0.717	0.720	0.722
Tomcat	0.685	0.664	0.666	0.669	0.670	0.672	0.674	0.677	0.682	0.688	0.692	0.691
Corefx	0.769	0.768	0.762	0.762	0.764	0.761	0.758	0.764	0.768	0.765	0.778	0.786
Django	0.753	0.756	0.756	0.757	0.757	0.757	0.761	0.761	0.764	0.766	0.766	0.763
Rails	0.656	0.618	0.622	0.629	0.639	0.646	0.648	0.653	0.658	0.664	0.669	0.670
Rust	0.687	0.682	0.683	0.688	0.689	0.694	0.695	0.694	0.700	0.697	0.704	0.707
Tensorflow	0.700	0.661	0.659	0.659	0.662	0.669	0.677	0.687	0.695	0.702	0.710	0.717
VScode	0.791	0.816	0.812	0.806	0.805	0.800	0.799	0.793	0.791	0.785	0.786	0.783
wp-Calypso	0.722	0.691	0.689	0.689	0.692	0.699	0.707	0.713	0.722	0.731	0.746	0.757
aveRank	7.07 (*)	7.71	8.71	8.71	8.29	8.50	7.43	5.93	4.71	4.29	3.14	3.50

Table 7. RQ1.1 & RQ1.2 – Average F1 score of JIT-SDP with HumLa at different amounts of human noiseacross 100 runs. Please refer to Table 2 for more description.

Dataset	Waiting time	HumLa at different amounts of human noise										
		0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Bracket	0.648	0.641	0.646	0.647	0.648	0.647	0.644	0.638	0.628	0.622	0.614	0.602
Broadleaf	0.557	0.645	0.640	0.628	0.616	0.601	0.584	0.566	0.552	0.535	0.520	0.503
Camel	0.680	0.700	0.698	0.694	0.693	0.690	0.688	0.683	0.679	0.674	0.663	0.655
Fabric8	0.648	0.661	0.661	0.660	0.658	0.655	0.651	0.647	0.638	0.627	0.612	0.596
jGroup	0.530	0.569	0.563	0.554	0.547	0.541	0.535	0.529	0.521	0.507	0.490	0.465
Nova	0.673	0.681	0.681	0.681	0.680	0.679	0.678	0.673	0.671	0.661	0.643	0.617
Tomcat	0.573	0.634	0.631	0.628	0.625	0.618	0.612	0.605	0.593	0.575	0.544	0.510
Corefx	0.591	0.586	0.590	0.585	0.582	0.583	0.575	0.573	0.568	0.563	0.555	0.546
Django	0.663	0.675	0.673	0.671	0.671	0.672	0.672	0.671	0.668	0.660	0.645	0.625
Rails	0.524	0.680	0.667	0.649	0.634	0.610	0.586	0.564	0.543	0.520	0.496	0.472
Rust	0.538	0.540	0.544	0.543	0.544	0.538	0.537	0.534	0.524	0.518	0.504	0.502
Tensorflow	0.689	0.721	0.719	0.718	0.718	0.716	0.712	0.703	0.692	0.676	0.656	0.635
VScode	0.459	0.452	0.447	0.442	0.435	0.433	0.426	0.414	0.405	0.396	0.387	0.373
wp-Calypso	0.475	0.583	0.584	0.579	0.574	0.560	0.543	0.520	0.494	0.456	0.401	0.346
aveRank	6.64 (*)	2.00	2.14	3.43	3.93	4.86	6.07	7.36	8.64	9.93	11.00	12.00

Table 8. RQ1.3 – Average G-Mean of JIT-SDP with HumLa at different amounts of human effort across 100 runs. The waiting time method is equivalent to HumLa at 0%-human effort and is chosen as the control method. The last row reports the statistical tests across datasets. Significant difference against the control method is highlighted in yellow (light gray). Smaller rankings represent better predictive performance for JIT-SDP when there is statistically significant difference.

Dataset	HumLa at different amounts of human effort										
	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%	0%
Bracket	0.639	0.639	0.640	0.642	0.642	0.641	0.642	0.643	0.644	0.644	0.643
Broadleaf	0.663	0.661	0.659	0.656	0.652	0.646	0.640	0.634	0.627	0.618	0.607
Camel	0.681	0.678	0.679	0.678	0.676	0.676	0.674	0.674	0.672	0.670	0.669
Fabric8	0.661	0.659	0.659	0.657	0.657	0.657	0.656	0.656	0.656	0.654	0.653
jGroup	0.600	0.598	0.598	0.596	0.590	0.588	0.586	0.583	0.577	0.572	0.568
Nova	0.688	0.688	0.688	0.688	0.688	0.688	0.687	0.687	0.686	0.685	0.682
Tomcat	0.638	0.637	0.637	0.636	0.637	0.634	0.633	0.629	0.627	0.622	0.613
Corefx	0.636	0.638	0.637	0.634	0.637	0.634	0.636	0.635	0.636	0.637	0.639
Django	0.698	0.697	0.697	0.697	0.696	0.695	0.695	0.695	0.694	0.693	0.690
Rails	0.623	0.625	0.625	0.623	0.619	0.616	0.607	0.595	0.584	0.576	0.562
Rust	0.586	0.588	0.592	0.591	0.592	0.592	0.589	0.586	0.580	0.585	0.584
Tensorflow	0.678	0.679	0.681	0.683	0.686	0.688	0.690	0.693	0.694	0.693	0.691
VScode	0.527	0.527	0.524	0.523	0.527	0.529	0.533	0.535	0.536	0.534	0.527
wp-Calypso	0.622	0.623	0.623	0.619	0.615	0.610	0.605	0.593	0.583	0.566	0.551
aveRank	4.04	3.96	4.11	5.36	4.82	6.36	6.54	6.75	7.21	7.86	9.00 (*)

Table 9. RQ1.3 – Average MCC of JIT-SDP with HumLa at different amounts of human effort across 100 runs. The waiting time method is equivalent to HumLa at 0%-human effort and is chosen as the control method. Please refer to Table 8 for more description.

Dataset	HumLa at different amounts of human effort										
	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%	0%
Bracket	0.297	0.298	0.299	0.302	0.302	0.301	0.301	0.302	0.304	0.303	0.300
Broadleaf	0.347	0.345	0.343	0.339	0.334	0.329	0.323	0.318	0.312	0.303	0.292
Camel	0.378	0.374	0.377	0.376	0.372	0.373	0.369	0.368	0.365	0.360	0.356
Fabric8	0.333	0.331	0.331	0.329	0.329	0.328	0.327	0.326	0.324	0.321	0.320
jGroup	0.242	0.241	0.238	0.233	0.226	0.220	0.217	0.208	0.201	0.195	0.193
Nova	0.391	0.391	0.391	0.391	0.391	0.390	0.389	0.388	0.387	0.384	0.380
Tomcat	0.309	0.308	0.309	0.308	0.309	0.307	0.305	0.299	0.295	0.290	0.282
Corefx	0.360	0.360	0.359	0.348	0.355	0.350	0.351	0.348	0.354	0.356	0.362
Django	0.430	0.427	0.427	0.427	0.424	0.423	0.422	0.423	0.421	0.419	0.413
Rails	0.296	0.295	0.292	0.285	0.277	0.272	0.258	0.243	0.233	0.228	0.214
Rust	0.248	0.250	0.256	0.255	0.258	0.253	0.251	0.247	0.244	0.250	0.250
Tensorflow	0.394	0.394	0.395	0.397	0.397	0.397	0.398	0.397	0.397	0.393	0.389
VScode	0.302	0.299	0.292	0.290	0.293	0.293	0.298	0.297	0.295	0.287	0.276
wp-Calypso	0.293	0.294	0.294	0.291	0.291	0.289	0.288	0.281	0.274	0.265	0.256
aveRank	3.36	3.71	3.86	4.79	4.57	5.71	6.29	7.21	7.93	8.86	9.71 (*)

Table 10. RQ1.3 – Average Recall 1 of JIT-SDP with HumLa at different amounts of human effort across 100 runs. The waiting time method is equivalent to HumLa at 0%-human effort and is chosen as the control method. Please refer to Table 8 for more description.

Dataset	HumLa at different amounts of human effort										
	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%	0%
Bracket	0.638	0.644	0.647	0.652	0.655	0.663	0.666	0.662	0.664	0.659	0.652
Broadleaf	0.605	0.597	0.590	0.578	0.568	0.552	0.537	0.525	0.509	0.491	0.470
Camel	0.736	0.734	0.737	0.735	0.732	0.734	0.728	0.723	0.719	0.710	0.700
Fabric8	0.662	0.665	0.665	0.663	0.663	0.665	0.660	0.656	0.653	0.646	0.639
jGroup	0.517	0.507	0.516	0.516	0.509	0.515	0.517	0.520	0.506	0.494	0.471
Nova	0.660	0.661	0.661	0.661	0.661	0.661	0.660	0.658	0.657	0.652	0.646
Tomcat	0.629	0.628	0.624	0.619	0.614	0.603	0.596	0.579	0.568	0.545	0.508
Corefx	0.480	0.483	0.481	0.484	0.485	0.482	0.485	0.485	0.483	0.483	0.485
Django	0.620	0.618	0.618	0.616	0.615	0.613	0.613	0.610	0.606	0.603	0.600
Rails	0.764	0.748	0.739	0.705	0.670	0.650	0.600	0.552	0.520	0.495	0.465
Rust	0.457	0.461	0.470	0.467	0.466	0.473	0.464	0.463	0.450	0.452	0.452
Tensorflow	0.800	0.798	0.794	0.788	0.779	0.771	0.760	0.742	0.725	0.701	0.683
VScode	0.328	0.328	0.325	0.325	0.329	0.333	0.340	0.344	0.348	0.348	0.347
wp-Calypso	0.513	0.519	0.526	0.520	0.503	0.486	0.472	0.445	0.422	0.388	0.363
aveRank	4.64	4.79	4.00	4.50	4.79	5.21	5.57	6.43	7.93	8.71	9.43 (*)

Table 11. RQ1.3 – Average Recall 0 of JIT-SDP with HumLa at different amounts of human effort across 100 runs. The waiting time method is equivalent to HumLa at 0%-human effort and is chosen as the control method. Please refer to Table 8 for more description.

Dataset	HumLa at different amounts of human effort										
	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%	0%
Bracket	0.654	0.649	0.647	0.645	0.643	0.633	0.630	0.637	0.636	0.640	0.645
Broadleaf	0.736	0.740	0.745	0.753	0.757	0.766	0.773	0.780	0.787	0.794	0.802
Camel	0.636	0.635	0.633	0.633	0.634	0.632	0.634	0.638	0.638	0.643	0.650
Fabric8	0.669	0.663	0.662	0.662	0.662	0.659	0.663	0.667	0.668	0.671	0.678
jGroup	0.716	0.724	0.712	0.708	0.707	0.696	0.690	0.679	0.684	0.691	0.710
Nova	0.727	0.725	0.725	0.725	0.725	0.725	0.724	0.725	0.725	0.727	0.729
Tomcat	0.671	0.672	0.676	0.680	0.687	0.695	0.700	0.710	0.718	0.735	0.762
Corefx	0.853	0.850	0.850	0.839	0.844	0.841	0.840	0.838	0.844	0.846	0.849
Django	0.799	0.798	0.798	0.800	0.799	0.799	0.799	0.802	0.804	0.805	0.803
Rails	0.519	0.535	0.543	0.571	0.597	0.613	0.648	0.680	0.699	0.717	0.730
Rust	0.772	0.770	0.769	0.770	0.774	0.763	0.769	0.767	0.774	0.778	0.778
Tensorflow	0.582	0.585	0.591	0.599	0.610	0.619	0.632	0.652	0.668	0.690	0.703
VScode	0.906	0.904	0.902	0.900	0.900	0.897	0.895	0.892	0.886	0.879	0.869
wp-Calypso	0.769	0.763	0.756	0.758	0.773	0.787	0.798	0.814	0.827	0.847	0.858
aveRank	5.79	6.36	7.36	7.32	6.89	7.79	7.36	6.36	4.71	3.43	2.64 (*)

Table 12. RQ1.3 – Average Precision of JIT-SDP with HumLa at different amounts of human effort across 100 runs. The waiting time method is equivalent to HumLa at 0%-human effort and is chosen as the control method. Please refer to Table 8 for more description.

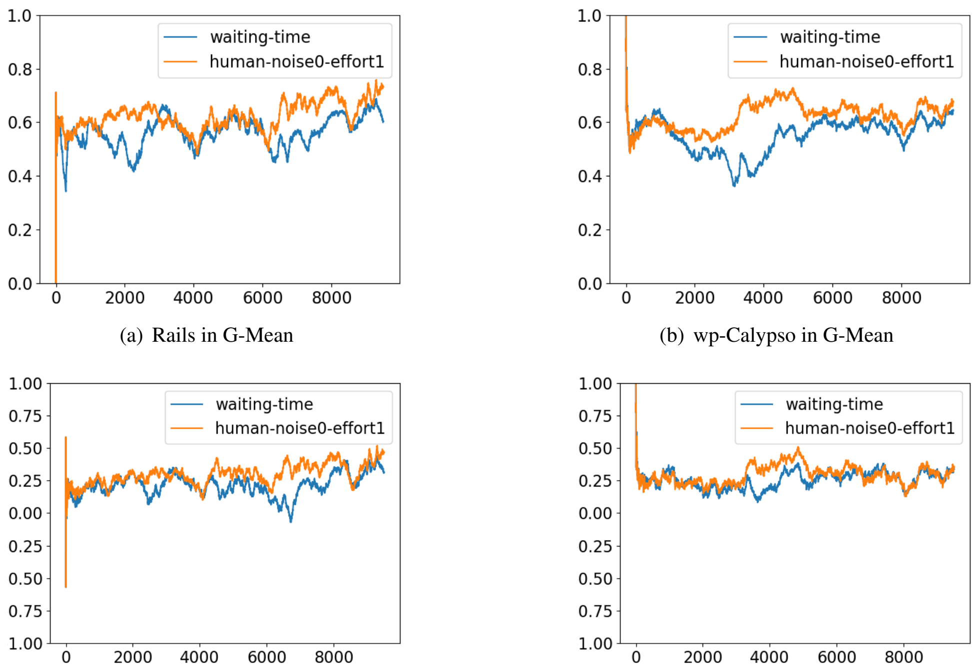
Dataset	HumLa at different amounts of human effort										
	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%	0%
Bracket	0.654	0.652	0.652	0.653	0.652	0.649	0.648	0.651	0.651	0.652	0.653
Broadleaf	0.701	0.702	0.703	0.705	0.705	0.707	0.708	0.709	0.710	0.710	0.710
Camel	0.673	0.671	0.671	0.671	0.671	0.670	0.670	0.671	0.671	0.671	0.672
Fabric8	0.668	0.666	0.665	0.664	0.664	0.663	0.663	0.664	0.664	0.664	0.666
jGroup	0.654	0.656	0.651	0.648	0.645	0.641	0.638	0.632	0.631	0.631	0.634
Nova	0.713	0.713	0.713	0.713	0.713	0.712	0.711	0.712	0.711	0.711	0.710
Tomcat	0.664	0.664	0.666	0.666	0.669	0.671	0.671	0.673	0.674	0.678	0.685
Corefx	0.768	0.767	0.767	0.756	0.762	0.759	0.758	0.756	0.763	0.764	0.769
Django	0.756	0.755	0.755	0.756	0.754	0.754	0.753	0.755	0.756	0.756	0.753
Rails	0.618	0.622	0.623	0.629	0.635	0.638	0.644	0.648	0.651	0.656	0.656
Rust	0.682	0.682	0.684	0.685	0.687	0.681	0.682	0.680	0.682	0.686	0.687
Tensorflow	0.661	0.662	0.664	0.666	0.670	0.673	0.677	0.683	0.689	0.696	0.700
VScode	0.816	0.811	0.805	0.802	0.803	0.801	0.805	0.801	0.798	0.793	0.791
wp-Calypso	0.691	0.689	0.687	0.687	0.693	0.699	0.703	0.708	0.712	0.719	0.722
aveRank	4.86	6.07	6.00	6.07	6.36	7.71	7.50	6.36	5.93	5.21	3.93 (*)

Table 13. RQ1.3 – Average F1 score of JIT-SDP with HumLa at different amounts of human effort across 100 runs. The waiting time method is equivalent to HumLa at 0%-human effort and is chosen as the control method. Please refer to Table 8 for more description.

Dataset	HumLa at different amounts of human effort										
	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%	0%
Bracket	0.641	0.643	0.645	0.648	0.649	0.652	0.653	0.652	0.654	0.652	0.648
Broadleaf	0.645	0.641	0.638	0.631	0.625	0.615	0.605	0.597	0.586	0.573	0.557
Camel	0.700	0.697	0.699	0.698	0.695	0.696	0.693	0.691	0.689	0.684	0.680
Fabric8	0.661	0.661	0.661	0.659	0.659	0.660	0.657	0.656	0.654	0.651	0.648
jGroup	0.569	0.564	0.568	0.566	0.560	0.561	0.561	0.560	0.551	0.543	0.530
Nova	0.681	0.682	0.681	0.682	0.682	0.681	0.681	0.680	0.679	0.677	0.673
Tomcat	0.634	0.633	0.631	0.629	0.627	0.622	0.618	0.610	0.604	0.593	0.573
Corefx	0.586	0.589	0.587	0.586	0.589	0.586	0.588	0.588	0.588	0.588	0.591
Django	0.675	0.673	0.674	0.672	0.672	0.670	0.671	0.670	0.668	0.667	0.663
Rails	0.680	0.675	0.671	0.657	0.641	0.631	0.607	0.579	0.560	0.545	0.524
Rust	0.540	0.543	0.550	0.548	0.549	0.551	0.545	0.543	0.533	0.538	0.538
Tensorflow	0.721	0.721	0.720	0.720	0.718	0.716	0.714	0.709	0.705	0.696	0.689
VScode	0.452	0.451	0.448	0.447	0.451	0.454	0.460	0.462	0.465	0.463	0.459
wp-Calypso	0.583	0.585	0.588	0.583	0.573	0.564	0.555	0.537	0.521	0.496	0.475
aveRank	3.86	3.64	3.93	4.93	5.07	5.57	6.14	7.07	7.71	8.57	9.50 (*)

2.1.2 Additional Result Plots for RQ1.1. This section provides additional plots including the continuous predictive performance of HumLa against the waiting time method throughout time steps for further insights for readers who are interested in a more detailed analysis of the results in performance.

Figure 1 shows continuous performance of HumLa against the waiting time method throughout test time steps on two dataset where HumLa led to substantially better predictive performance than the waiting time method. We can see that HumLa led to prolonged periods of time with consistently and considerably better performance in these two datasets, even though the performance was not always better throughout all time steps.

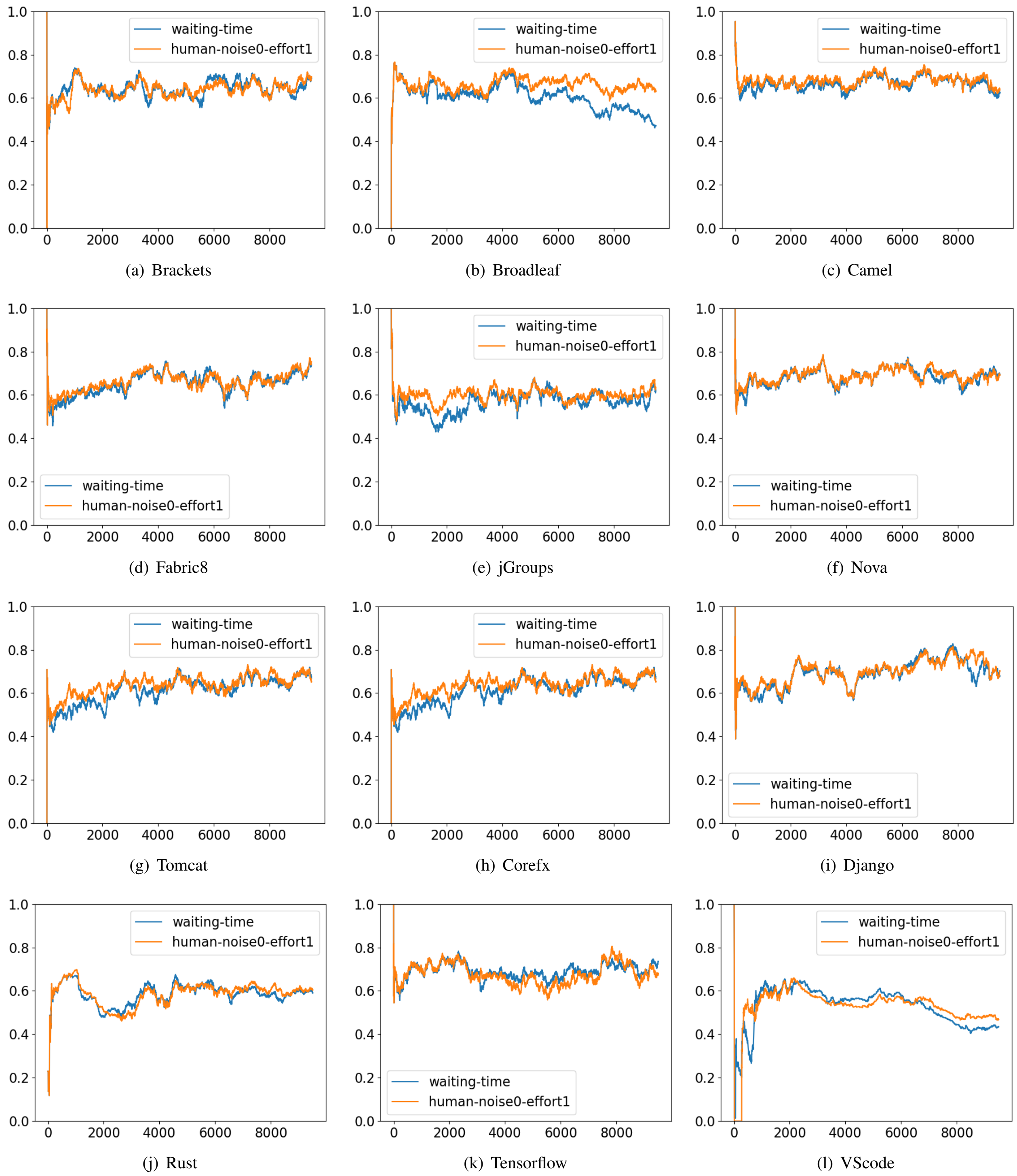


(a) Rails in G-Mean (b) wp-Calypso in G-Mean

(c) Rails in MCC (d) wp-Calypso in MCC

Fig. 1. RQ1.1 – Continuous performance comparison throughout time between HumLa at the default setup (orange lines) and the waiting time method (blue lines) on two representative dataset where HumLa provides significant benefit to the performance compared with the waiting time method in sustantially large magnitudes.

Plots of additional datasets are presented in Figure 2 and Figure 3 of the supplementary material, illustrating the trends in G-Mean and MCC, respectively. These plots provide further insights for readers who are interested in a more detailed analysis of the results, which especially illustrates the cases when HumLa does not bring large improvement in performance.



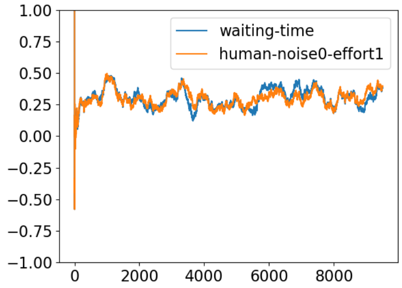
(a) Brackets (b) Broadleaf (c) Camel

(d) Fabric8 (e) jGroups (f) Nova

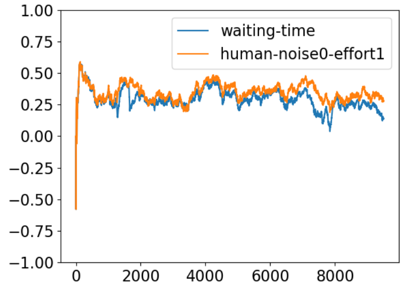
(g) Tomcat (h) Corefx (i) Django

(j) Rust (k) Tensorflow (l) VScode

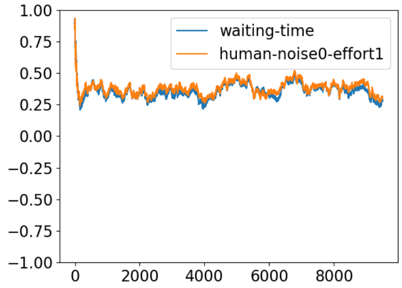
Fig. 2. RQ1.1 – Continuous predictive performance in terms of average G-Mean across 100 runs throughout test between HumLa at the default setup (orange lines) and the waiting time method (blue lines).



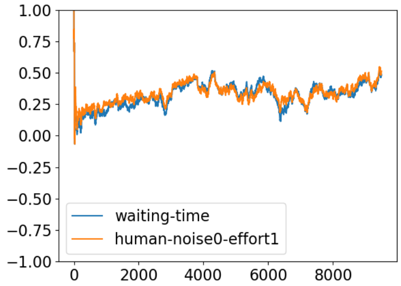
(a) Brackets



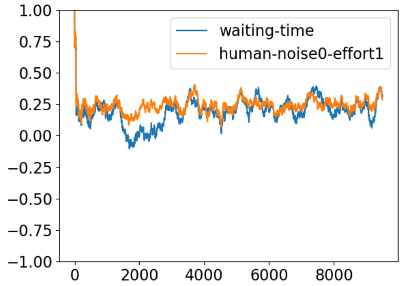
(b) Broadleaf



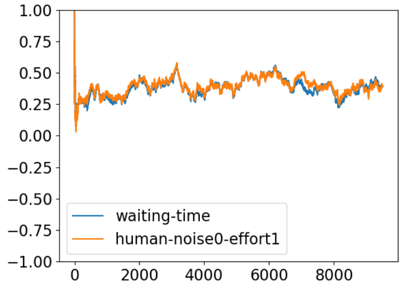
(c) Camel



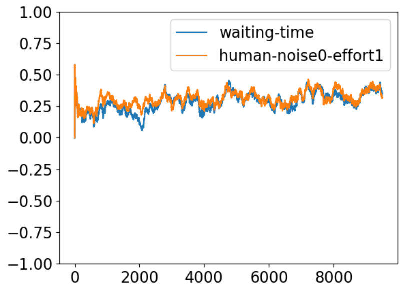
(d) Fabric8



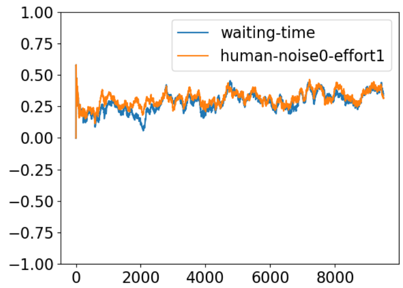
(e) jGroups



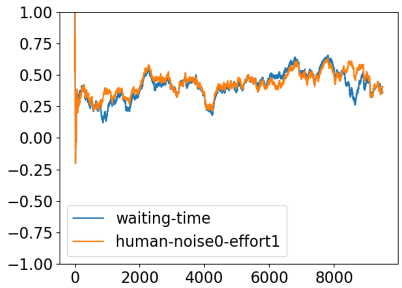
(f) Nova



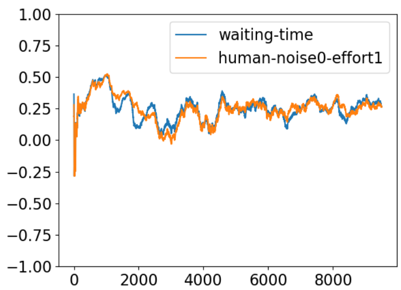
(g) Tomcat



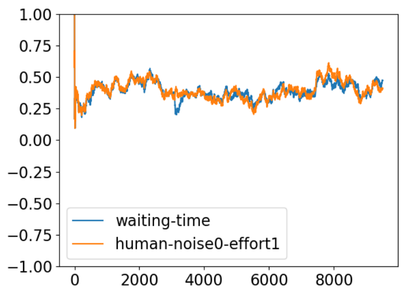
(h) Corefx



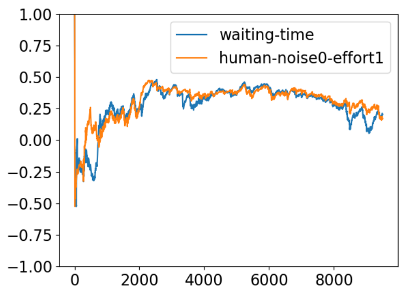
(i) Django



(j) Rust



(k) Tensorflow



(l) VScode

Fig. 3. RQ1.1 – Continuous predictive performance in terms of average MCC across 100 runs throughout test between HumLa at the default setup (orange lines) and the waiting time method (blue lines).

2.1.3 Additional Result Plot for RQ1.3. Figure 4 of this supplementary material shows the relationship between the cumulative code churn and the human labeling percentage on the investigated datasets. We can see that higher human labeling percentage almost always accounts for larger value of the cumulative code churn, showing a good correlation between these two metrics. Therefore, given a project for which practitioners may be interested in creating a JIT-SDP predictive model, reducing the inspection rate is really likely to correlate with a decrease in churn for this project.

Table 14 contains the code churn values for all datasets. We can see that, for example, HumLa at 40%- and 60%-human effort would reduce around up 40% and 60% code churns, respectively, being consistent to the human labeling percentages.

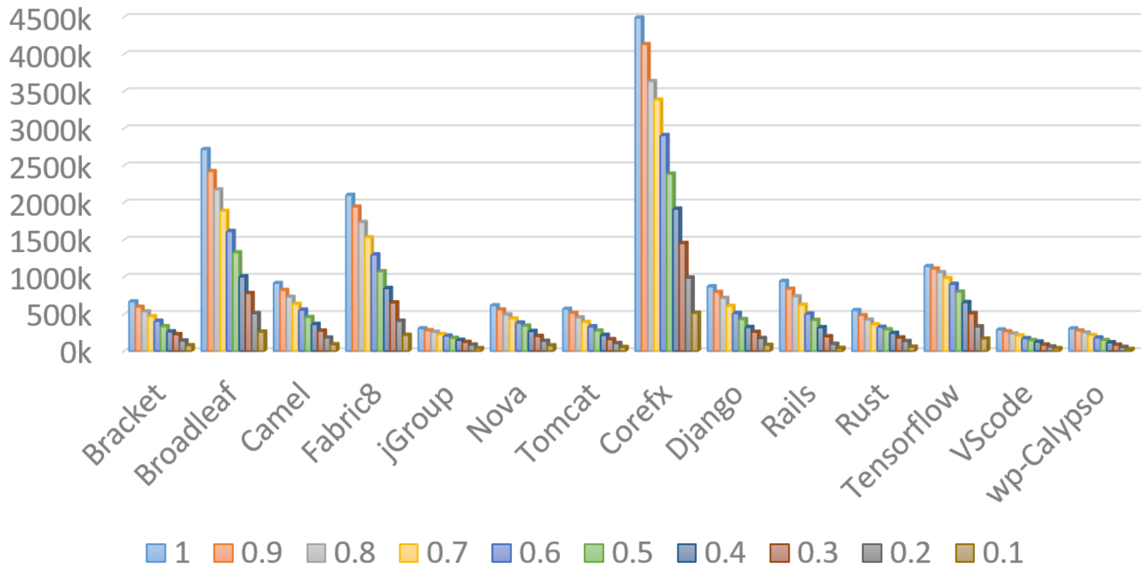


Fig. 4. RQ1.3 – Relationship between the human labeling percentage and the cumulative code churn for HumLa at varying amounts of human effort on each dataset.

Table 14. RQ2.2 – Saved human effort of the proposed ECo-HumLa against HumLa at different amounts of human effort in terms of the human labeling percentage and the cumulative code churn (in kilo).

Dataset	Human effort (in kilo) of HumLa					Eco-HumLa		Human effort (in kilo) of HumLa				
	100%	90%	80%	70%	60%	auto	human%	50%	40%	30%	20%	10%
Bracket	664.8 [b]	593.2 [b]	530.4 [b]	466.0 [b]	403.9 [m]	371.0	55.51%	331.5 [-m]	261.1 [-b]	223.2 [-b]	137.9 [-b]	74.3 [-b]
Broadleaf	2715.4 [b]	2418.2 [b]	2171.9 [*]	1884.0 [-b]	1612.6 [-b]	2162.6	46.60%	1326.4 [-b]	1002.0 [-b]	777.2 [-b]	506.9 [-b]	258.2 [-b]
Camel	913.3 [b]	818.0 [b]	727.8 [b]	633.0 [b]	553.6 [-m]	578.2	54.57%	455.0 [-b]	360.5 [-b]	271.8 [-b]	177.9 [-b]	87.6 [-b]
Fabric8	2099.6 [b]	1941.0 [b]	1733.7 [b]	1528.7 [b]	1297.5 [m]	1165.2	46.85%	1071.2 [-s]	844.6 [-b]	651.0 [-b]	403.9 [-b]	214.5 [-b]
jGroup	302.8 [b]	276.2 [b]	255.3 [b]	224.4 [b]	203.8 [-*]	200.0	45.55%	174.8 [-b]	147.8 [-b]	118.6 [-b]	83.4 [-b]	35.6 [-b]
Nova	612.7 [b]	555.1 [b]	488.5 [b]	434.7 [b]	380.4 [m]	332.7	62.09%	338.5 [*]	267.1 [-b]	199.3 [-b]	134.2 [-b]	73.1 [-b]
Tomcat	564.9 [b]	510.0 [b]	450.8 [b]	386.8 [m]	329.3 [-b]	373.2	58.01%	273.5 [-b]	214.7 [-b]	156.0 [-b]	104.1 [-b]	50.5 [-b]
Corefx	4486.7 [b]	4129.2 [b]	3631.0 [b]	3382.0 [-*]	2904.6 [-b]	3305.1	51.95%	2384.0 [-b]	1913.4 [-b]	1450.3 [-b]	985.9 [-b]	512.4 [-b]
Django	867.6 [b]	793.3 [b]	711.8 [b]	605.3 [b]	509.1 [-*]	499.8	71.28%	426.5 [-b]	321.5 [-b]	255.6 [-b]	171.3 [-b]	79.6 [-b]
Rails	940.6 [b]	834.8 [b]	733.1 [b]	619.4 [b]	500.5 [b]	359.1	47.07%	416.2 [m]	317.9 [-s]	195.5 [-b]	93.6 [-b]	38.5 [-b]
Rust	547.8 [b]	479.8 [b]	418.9 [b]	356.6 [b]	323.1 [b]	234.2	40.73%	289.7 [m]	241.7 [*]	179.0 [-m]	130.7 [-b]	55.0 [-b]
Tensorflow	1140.3 [b]	1106.2 [b]	1058.1 [b]	979.3 [m]	902.2 [*]	900.4	55.94%	796.3 [-b]	655.1 [-b]	506.2 [-b]	328.5 [-b]	164.5 [-b]
VScode	286.1 [b]	262.6 [b]	232.6 [b]	206.7 [*]	170.2 [-b]	203.4	48.64%	144.4 [-b]	123.8 [-b]	83.4 [-b]	57.5 [-b]	35.6 [-b]
wp-Calypso	301.9 [b]	273.5 [b]	245.8 [b]	211.6 [b]	178.9 [-b]	187.8	51.41%	145.3 [-b]	114.4 [-b]	82.6 [-b]	52.1 [-b]	24.0 [-b]
median	766.2	693.3	621.1	535.7	452.2	372.1	51.68%	377.3	292.5	211.3	136.1	73.7

The proposed Eco-HumLa was the control method. A12 effect size [6] was performed for each dataset to rule out insignificant differences against the control method. Symbols [*], [s], [m] and [b] denote insignificant (<0.56), small (≥0.56), medium (≥0.64), and large (≥0.71) effect size against the control method, respectively. Presence / absence of the sign "-" in A12 means that the corresponding approach was worse/better than the control method.

2.2 RQ2: JIT-SDP with Eco-HumLa

This section reports additional results in performance relating to RQ2.1 in various evaluation metrics. The full experimental results to answer RQ2.2 and RQ2.3 are available in the main paper and thus there is no need to provide additional details in this supplementary material.

Table 15. RQ2.1 – Average G-Mean of JIT-SDP with ECo-HumLa vs HumLa at different amounts of human effort across 100 runs. HumLa at 0%-human effort is equivalent to the waiting time method. HumLa at 100%-human effort is chosen as the control method. Significant difference against the control method is highlighted in yellow (light gray). Smaller rankings represent better predictive performance for JIT-SDP when there is significant difference.

Dataset	ECo-HumLa vs HumLa at different amounts of human effort											
	100%	90%	80%	70%	60%	50%	ECo-HumLa	40%	30%	20%	10%	0%
Bracket	0.639	0.639	0.640	0.642	0.642	0.641	0.643	0.642	0.643	0.644	0.644	0.643
Broadleaf	0.663	0.661	0.659	0.656	0.652	0.646	0.640	0.640	0.634	0.627	0.618	0.607
Camel	0.681	0.678	0.679	0.678	0.676	0.676	0.678	0.674	0.674	0.672	0.670	0.669
Fabric8	0.661	0.659	0.659	0.657	0.657	0.657	0.657	0.656	0.656	0.656	0.654	0.653
jGroup	0.600	0.598	0.598	0.596	0.590	0.588	0.587	0.586	0.583	0.577	0.572	0.568
Nova	0.688	0.688	0.688	0.688	0.688	0.688	0.687	0.687	0.687	0.686	0.685	0.682
Tomcat	0.638	0.637	0.637	0.636	0.637	0.634	0.630	0.633	0.629	0.627	0.622	0.613
Corefx	0.636	0.638	0.637	0.634	0.637	0.634	0.631	0.636	0.635	0.636	0.637	0.639
Django	0.698	0.697	0.697	0.697	0.696	0.695	0.695	0.695	0.695	0.694	0.693	0.690
Rails	0.623	0.625	0.625	0.623	0.619	0.616	0.605	0.607	0.595	0.584	0.576	0.562
Rust	0.586	0.588	0.592	0.591	0.592	0.592	0.584	0.589	0.586	0.580	0.585	0.584
Tensorflow	0.678	0.679	0.681	0.683	0.686	0.688	0.701	0.690	0.693	0.694	0.693	0.691
VScode	0.527	0.527	0.524	0.523	0.527	0.529	0.534	0.533	0.535	0.536	0.534	0.527
wp-Calypso	0.622	0.623	0.623	0.619	0.615	0.610	0.601	0.605	0.593	0.583	0.566	0.551
aveRank	4.29 (*)	4.00	4.36	5.57	5.21	6.71	6.93	7.07	7.36	8.00	8.57	9.93

Table 16. RQ2.1 – Average MCC of JIT-SDP with ECo-HumLa vs HumLa at different amounts of human effort across 100 runs. HumLa at 0%-human effort is equivalent to the waiting time method. HumLa at 100%-human effort is chosen as the control method. Please refer to Table 15 for more description.

Dataset	ECo-HumLa vs HumLa at different amounts of human effort											
	100%	90%	80%	70%	60%	50%	ECo-HumLa	40%	30%	20%	10%	0%
Bracket	0.297	0.298	0.299	0.302	0.302	0.301	0.303	0.301	0.302	0.304	0.303	0.300
Broadleaf	0.347	0.345	0.343	0.339	0.334	0.329	0.327	0.323	0.318	0.312	0.303	0.292
Camel	0.378	0.374	0.377	0.376	0.372	0.373	0.374	0.369	0.368	0.365	0.360	0.356
Fabric8	0.333	0.331	0.331	0.329	0.329	0.328	0.329	0.327	0.326	0.324	0.321	0.320
jGroup	0.242	0.241	0.238	0.233	0.226	0.220	0.220	0.217	0.208	0.201	0.195	0.193
Nova	0.391	0.391	0.391	0.391	0.391	0.390	0.389	0.389	0.388	0.387	0.384	0.380
Tomcat	0.309	0.308	0.309	0.308	0.309	0.307	0.305	0.305	0.299	0.295	0.290	0.282
Corefx	0.360	0.360	0.359	0.348	0.355	0.350	0.350	0.351	0.348	0.354	0.356	0.362
Django	0.430	0.427	0.427	0.427	0.424	0.423	0.425	0.422	0.423	0.421	0.419	0.413
Rails	0.296	0.295	0.292	0.285	0.277	0.272	0.254	0.258	0.243	0.233	0.228	0.214
Rust	0.248	0.250	0.256	0.255	0.258	0.253	0.251	0.251	0.247	0.244	0.250	0.250
Tensorflow	0.394	0.394	0.395	0.397	0.397	0.397	0.408	0.398	0.397	0.397	0.393	0.389
VScode	0.302	0.299	0.292	0.290	0.293	0.293	0.288	0.298	0.297	0.295	0.287	0.276
wp-Calypso	0.293	0.294	0.294	0.291	0.291	0.289	0.285	0.288	0.281	0.274	0.265	0.256
ave-rank	3.57 (*)	3.93	4.00	5.07	4.93	6.07	6.21	6.93	8.14	8.71	9.79	10.64

Table 17. RQ2.1 – Average Recall 1 of JIT-SDP with ECo-HumLa vs HumLa at different amounts of human effort across 100 runs. HumLa at 0%-human effort is equivalent to the waiting time method. HumLa at 100%-human effort is chosen as the control method. Please refer to Table 15 for more description.

Dataset	ECo-HumLa vs HumLa at different amounts of human effort											
	100%	90%	80%	70%	60%	50%	40%	30%	ECo-HumLa	20%	10%	0%
Bracket	0.638	0.644	0.647	0.652	0.655	0.663	0.666	0.662	0.645	0.664	0.659	0.652
Broadleaf	0.605	0.597	0.590	0.578	0.568	0.552	0.537	0.525	0.528	0.509	0.491	0.470
Camel	0.736	0.734	0.737	0.735	0.732	0.734	0.728	0.723	0.724	0.719	0.710	0.700
Fabric8	0.662	0.665	0.665	0.663	0.663	0.665	0.660	0.656	0.652	0.653	0.646	0.639
jGroup	0.517	0.507	0.516	0.516	0.509	0.515	0.517	0.520	0.505	0.506	0.494	0.471
Nova	0.660	0.661	0.661	0.661	0.661	0.661	0.660	0.658	0.662	0.657	0.652	0.646
Tomcat	0.629	0.628	0.624	0.619	0.614	0.603	0.596	0.579	0.589	0.568	0.545	0.508
Corefx	0.480	0.483	0.481	0.484	0.485	0.482	0.485	0.485	0.472	0.483	0.483	0.485
Django	0.620	0.618	0.618	0.616	0.615	0.613	0.613	0.610	0.604	0.606	0.603	0.600
Rails	0.764	0.748	0.739	0.705	0.670	0.650	0.600	0.552	0.588	0.520	0.495	0.465
Rust	0.457	0.461	0.470	0.467	0.466	0.473	0.464	0.463	0.444	0.450	0.452	0.452
Tensorflow	0.800	0.798	0.794	0.788	0.779	0.771	0.760	0.742	0.726	0.725	0.701	0.683
VScode	0.328	0.328	0.325	0.325	0.329	0.333	0.340	0.344	0.344	0.348	0.348	0.347
wp-Calypso	0.513	0.519	0.526	0.520	0.503	0.486	0.472	0.445	0.452	0.422	0.388	0.363
aveRank	4.86 (*)	5.00	4.14	4.64	4.93	5.36	5.71	6.93	8.43	8.43	9.43	10.14

Table 18. RQ2.1 – Average Recall 0 of JIT-SDP with ECo-HumLa vs HumLa at different amounts of human effort across 100 runs. HumLa at 0%-human effort is equivalent to the waiting time method. HumLa at 100%-human effort is chosen as the control method. Please refer to Table 15 for more description.

Dataset	ECo-HumLa vs HumLa at different amounts of human effort											
	100%	90%	80%	70%	60%	50%	40%	30%	20%	ECo-HumLa	10%	0%
Bracket	0.654	0.649	0.647	0.645	0.643	0.633	0.630	0.637	0.636	0.654	0.640	0.645
Broadleaf	0.736	0.740	0.745	0.753	0.757	0.766	0.773	0.780	0.787	0.785	0.794	0.802
Camel	0.636	0.635	0.633	0.633	0.634	0.632	0.634	0.638	0.638	0.643	0.643	0.650
Fabric8	0.669	0.663	0.662	0.662	0.662	0.659	0.663	0.667	0.668	0.673	0.671	0.678
jGroup	0.716	0.724	0.712	0.708	0.707	0.696	0.690	0.679	0.684	0.705	0.691	0.710
Nova	0.727	0.725	0.725	0.725	0.725	0.725	0.724	0.725	0.725	0.723	0.727	0.729
Tomcat	0.671	0.672	0.676	0.680	0.687	0.695	0.700	0.710	0.718	0.705	0.735	0.762
Corefx	0.853	0.850	0.850	0.839	0.844	0.841	0.840	0.838	0.844	0.850	0.846	0.849
Django	0.799	0.798	0.798	0.800	0.799	0.799	0.799	0.802	0.804	0.810	0.805	0.803
Rails	0.519	0.535	0.543	0.571	0.597	0.613	0.648	0.680	0.699	0.656	0.717	0.730
Rust	0.772	0.770	0.769	0.770	0.774	0.763	0.769	0.767	0.774	0.786	0.778	0.778
Tensorflow	0.582	0.585	0.591	0.599	0.610	0.619	0.632	0.652	0.668	0.680	0.690	0.703
VScode	0.906	0.904	0.902	0.900	0.900	0.897	0.895	0.892	0.886	0.882	0.879	0.869
wp-Calypso	0.769	0.763	0.756	0.758	0.773	0.787	0.798	0.814	0.827	0.813	0.847	0.858
aveRank	6.43 (*)	7.07	8.07	8.11	7.68	8.64	8.21	7.00	5.29	4.57	3.93	3.00

Table 19. RQ2.1 – Average Precision of JIT-SDP with ECo-HumLa vs HumLa at different amounts of human effort across 100 runs. HumLa at 0%-human effort is equivalent to the waiting time method. HumLa at 100%-human effort is chosen as the control method. Please refer to Table 15 for more description.

Dataset	ECo-HumLa vs HumLa at different amounts of human effort											
	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%	0%	ECo-HumLa
Bracket	0.654	0.652	0.652	0.653	0.652	0.649	0.648	0.651	0.651	0.652	0.653	0.657
Broadleaf	0.701	0.702	0.703	0.705	0.705	0.707	0.708	0.709	0.710	0.710	0.710	0.715
Camel	0.673	0.671	0.671	0.671	0.671	0.670	0.670	0.671	0.671	0.671	0.672	0.675
Fabric8	0.668	0.666	0.665	0.664	0.664	0.663	0.663	0.664	0.664	0.664	0.666	0.668
jGroup	0.654	0.656	0.651	0.648	0.645	0.641	0.638	0.632	0.631	0.631	0.634	0.643
Nova	0.713	0.713	0.713	0.713	0.713	0.712	0.711	0.712	0.711	0.711	0.710	0.712
Tomcat	0.664	0.664	0.666	0.666	0.669	0.671	0.671	0.673	0.674	0.678	0.685	0.674
Corefx	0.768	0.767	0.767	0.756	0.762	0.759	0.758	0.756	0.763	0.764	0.769	0.763
Django	0.756	0.755	0.755	0.756	0.754	0.754	0.753	0.755	0.756	0.756	0.753	0.761
Rails	0.618	0.622	0.623	0.629	0.635	0.638	0.644	0.648	0.651	0.656	0.656	0.645
Rust	0.682	0.682	0.684	0.685	0.687	0.681	0.682	0.680	0.682	0.686	0.687	0.688
Tensorflow	0.661	0.662	0.664	0.666	0.670	0.673	0.677	0.683	0.689	0.696	0.700	0.696
VScode	0.816	0.811	0.805	0.802	0.803	0.801	0.805	0.801	0.798	0.793	0.791	0.796
wp-Calypso	0.691	0.689	0.687	0.687	0.693	0.699	0.703	0.708	0.712	0.719	0.722	0.710
aveRank	5.50 (*)	6.79	6.71	6.86	7.14	8.57	8.43	7.21	6.71	5.86	4.57	3.64

Table 20. RQ2.1 – Average F1 Score of JIT-SDP with ECo-HumLa vs HumLa at different amounts of human effort across 100 runs. HumLa at 0%-human effort is equivalent to the waiting time method. HumLa at 100%-human effort is chosen as the control method. Please refer to Table 15 for more description.

Dataset	ECo-HumLa vs HumLa at different amounts of human effort											
	100%	90%	80%	70%	60%	50%	40%	30%	ECo-HumLa	20%	10%	0%
Bracket	0.641	0.643	0.645	0.648	0.649	0.652	0.653	0.652	0.646	0.654	0.652	0.648
Broadleaf	0.645	0.641	0.638	0.631	0.625	0.615	0.605	0.597	0.602	0.586	0.573	0.557
Camel	0.700	0.697	0.699	0.698	0.695	0.696	0.693	0.691	0.694	0.689	0.684	0.680
Fabric8	0.661	0.661	0.661	0.659	0.659	0.660	0.657	0.656	0.655	0.654	0.651	0.648
jGroup	0.569	0.564	0.568	0.566	0.560	0.561	0.561	0.560	0.557	0.551	0.543	0.530
Nova	0.681	0.682	0.681	0.682	0.682	0.681	0.681	0.680	0.681	0.679	0.677	0.673
Tomcat	0.634	0.633	0.631	0.629	0.627	0.622	0.618	0.610	0.614	0.604	0.593	0.573
Corefx	0.586	0.589	0.587	0.586	0.589	0.586	0.588	0.588	0.579	0.588	0.588	0.591
Django	0.675	0.673	0.674	0.672	0.672	0.670	0.671	0.670	0.668	0.668	0.667	0.663
Rails	0.680	0.675	0.671	0.657	0.641	0.631	0.607	0.579	0.601	0.560	0.545	0.524
Rust	0.540	0.543	0.550	0.548	0.549	0.551	0.545	0.543	0.535	0.533	0.538	0.538
Tensorflow	0.721	0.721	0.720	0.720	0.718	0.716	0.714	0.709	0.709	0.705	0.696	0.689
VScode	0.452	0.451	0.448	0.447	0.451	0.454	0.460	0.462	0.471	0.465	0.463	0.459
wp-Calypso	0.583	0.585	0.588	0.583	0.573	0.564	0.555	0.537	0.546	0.521	0.496	0.475
aveRank	4.00 (*)	3.79	4.07	5.00	5.14	5.64	6.36	7.57	8.29	8.50	9.36	10.29

REFERENCES

[1] George G. Cabral, Leandro L. Minku, Emad Shihab, and Suhaib Mujahid. 2019. Class Imbalance Evolution and Verification Latency in Just-in-Time Software Defect Prediction. In *International Conference on Software Engineering*. Monteal, Canada, 666–676.

[2] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. 2013. A Large-Scale Empirical Study of Just-in-Time Quality Assurance. *IEEE Transactions on Software Engineering* 39, 6 (2013), 757–773.

[3] Nachiappan Nagappan and Thomas Ball. 2005. Use of Relative Code Churn Measures to Predict System Defect Density. In *International Conference on Software Engineering (ICSE)*. 284–292.

[4] Liyan Song, Shuxian Li, Leandro L. Minku, and Xin Yao. 2022. A Novel Data Stream Learning Approach to Tackle One-Sided Label Noise From Verification Latency. In *International Joint Conference on Neural Networks (IJCNN)*. 1–8.

[5] Liyan Song and Leandro L. Minku. 2023. A Procedure to Continuously Evaluate Predictive Performance of Just-In-Time Software Defect Prediction Models During Software Development. *IEEE Transactions on Software Engineering* 49, 2 (2023), 646–666. https://doi.org/10.1109/TSE.2022.3158831

[6] Chadd Williams and Jaime Spacco. 2008. SZZ Revisited: Verifying When Changes Induce Fixes. In *Proceedings of the 2008 Workshop on Defects in Large Software Systems*. 32–36.

[7] Xingguang Yang, Huiqun Yu, Guisheng Fan, Kai Shi, Liqiong Chen, and Emiliano Tramontana. 2019. Local versus Global Models for Just-In-Time Software Defect Prediction. *Scientific Programming* 2019 (2019), 1–13.

Received 2023-02-02; accepted 2023-07-27