A HYPER-PARAMETERS USED BY DIEMPH

For most hyper-parameters, we use the default values coming with the models. Specifically, for the models with the JTrans architecture, we use a learning-rate of $1e^{-5}$, a batch size of 64, and finetune the model for 10 epochs. For the models with the Trex architecture, we use a learning-rate of $1e^{-5}$, following the default setup of the Trex model. To accelerate the training process, we enlarge the batch size of Trex from 32 to 128 by distributing the training program on 4 GPUs. The original Trex model does not specify the epochs for training. We thus finetune the Trex models for 10 epochs on the BinKit and the How-Solve datasets. Given the extremely large size of the BinaryCorps-3M dataset, we finetune the Trex models on it for 50k steps, which relates to around 4 epochs and 15 hours training time. We can see from Fig. 10 the training converges on all three datasets with our hyper-parameters. Note that for each model/dataset setup, we use the same set of hyper-parameters to finetune both the original model and the model enhanced by DiEmph.

B PERFORMANCE OF DIEMPH IN OTHER METRICS

We show the effectiveness of DiEmph in terms of PR@5, PR@10, and MRR in Table 2, Table 3, and Table 4, respectively. We can see that models enhanced by DiEmph consistently achieve better performance in terms of all these metrics.

C EFFECTS ON IN-DISTRIBUTION DATA

Detailed results are shown in Table 5. We can see that in most cases, DiEmph slightly improve models' performance on average. For the Trex model trained with the BinaryCorp-3M dataset, the performance slightly degrades after applying DiEmph. As discussed in Section4.2, that is because the dataset contains complex control-flow features that goes beyond the capability of the Trex model. In these cases, the semantically unimportant instructions may indeed be critical for achieving good performance. Note that for models trained with the BinKit dataset, their performance on the in-distribution dataset is slightly lower than that on the out-of-distribution dataset. We speculate that is because the dataset is relatively small and not diverse such that the models overfit to some simple patterns that happens to exist in the out-of-distribution dataset.

D TIME EFFICIENCY OF DIEMPH

We record the time for the overall run time and the most time-consuming parts. The results are shown in Table 6. The first column shows the models, in the form of $ModelArch_{dataset}$. The 2nd–4th columns show the time for KDE, then classification importance analysis, and the overall run time. All numbers are wall-clock time in minutes. We can see that on average, DiEmph takes around 29 minutes to analyze one model with 200 sampled functions. Computing the classification importance for each instruction is most time-consuming, as the algorithm creates around 100 mutants of a given binary function and invokes the model to encode all the mutants. The time consumption of DiEmph is acceptable since it is a one-time effort.

Received 2023-02-16; accepted 2023-05-03

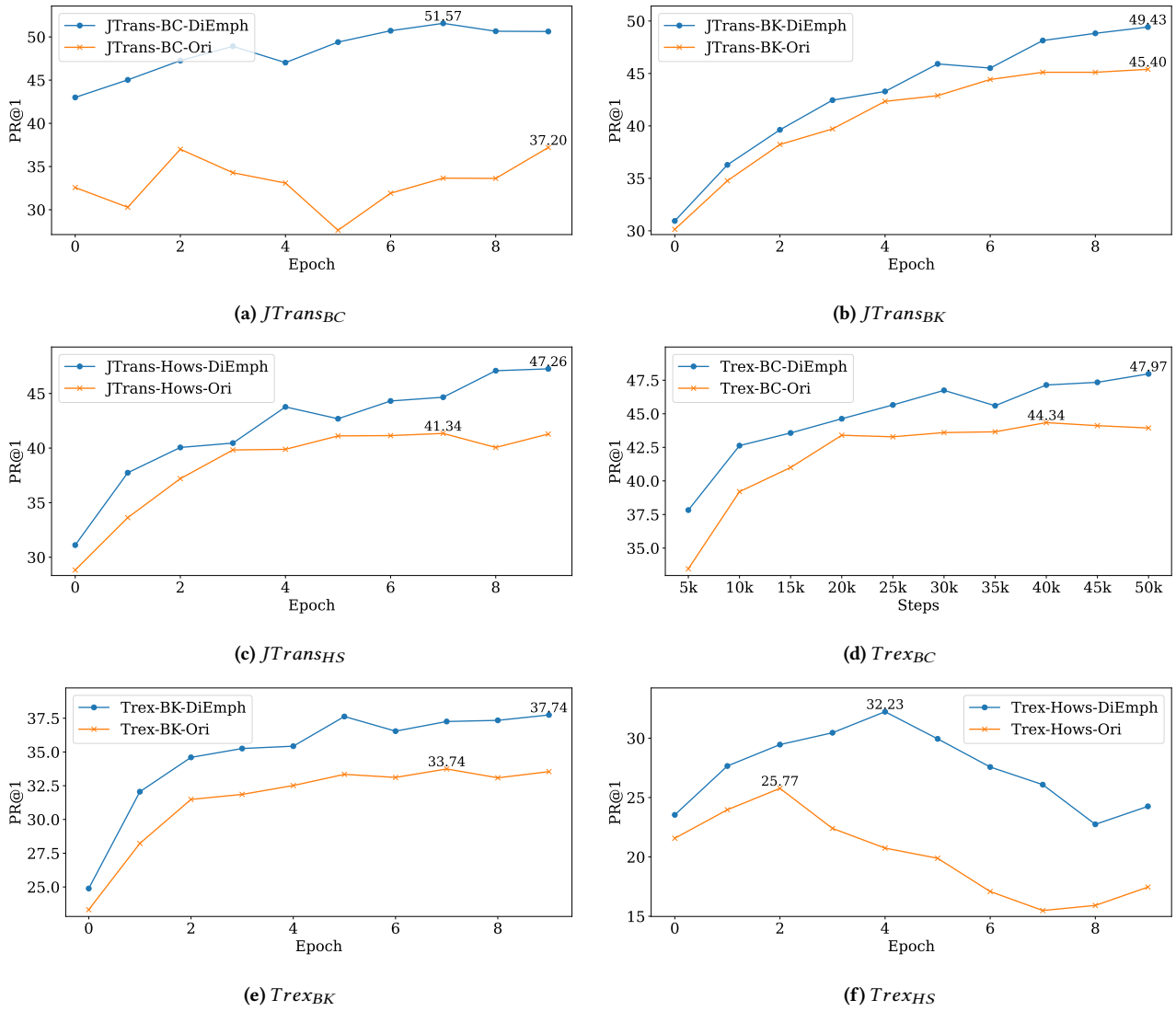

Figure 10: How DiEmph improves the performance of models across different training steps. Each sub-figure represents the training curves of one model/dataset setup. For each setup, the PR@1 of the best-performing checkpoints are noted in the figure. We can observe that DiEmph consistently improves the performance of models in the whole training procedural.

Table 2: Performance (PR@5) improvement on the out-of-distribution dataset. The first column lists the name of binary programs. The model setups are listed in the first row. Each model is denoted with its architecture and the training dataset, in the form of $ModelArch_{dataset}$. *BC*, *BK*, and *HS* denote BinaryCorp-3M, Binkit, and How-solve, respectively. In the second row, for each model, Ori. means the PR@5 for the original model, DiEmph means the PR@5 for the model improved by DiEmph, and Impr. means the improvement achieved by applying DiEmph to the model.

Programs	$JTrans_{BC}$			$JTrans_{BK}$			$JTrans_{HS}$			$Trex_{BC}$			$Trex_{BK}$			$Trex_{HS}$		
	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.
Curl	72.4	84.2	11.8	74.6	77.6	3.0	72.0	77.6	5.6	74.4	76.4	2.0	59.0	60.4	1.4	52.6	61.2	8.6
Binutils	66.4	78.0	11.6	73.2	76.4	3.2	63.4	68.2	4.8	63.4	68.0	4.6	55.4	61.2	5.8	46.4	54.6	8.2
Coreutils	53.0	62.0	9.0	59.6	57.4	-2.2	53.4	58.4	5.0	51.0	59.0	8.0	41.4	48.6	7.2	42.8	46.4	3.6
ImageMagick	47.0	67.6	20.6	69.0	70.6	1.6	52.0	67.6	15.6	66.2	71.2	5.0	46.0	54.6	8.6	42.8	54.0	11.2
SQLite	65.2	82.6	17.4	74.6	79.0	4.4	64.4	78.4	14.0	83.6	82.8	-0.8	64.8	71.4	6.6	57.6	66.8	9.2
OpenSSL	77.0	83.2	6.2	69.8	73.2	3.4	81.0	81.6	0.6	69.4	72.2	2.8	52.2	57.8	5.6	52.8	60.6	7.8
Putty	60.2	68.8	8.6	61.8	64.4	2.6	66.0	68.6	2.6	61.8	58.4	-3.4	53.6	61.0	7.4	48.6	54.2	5.6
Average	**63.0**	**75.2**	**12.2**	**68.9**	**71.2**	**2.3**	**64.6**	**71.5**	**6.9**	**67.1**	**69.7**	**2.6**	**53.2**	**59.3**	**6.1**	**49.1**	**56.8**	**7.7**

Table 3: Performance (PR@10) improvement on the out-of-distribution dataset. Columns and rows can be interpreted similarly to Table 2.

Programs	$JTrans_{BC}$			$JTrans_{BK}$			$JTrans_{HS}$			$Trex_{BC}$			$Trex_{BK}$			$Trex_{HS}$		
	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.
Curl	81.0	89.6	8.6	83.4	86.6	3.2	81.8	85.6	3.8	81.0	82.8	1.8	68.2	71.8	3.6	65.6	71.4	5.8
Binutils	75.8	85.8	10.0	80.8	83.6	2.8	73.4	78.6	5.2	75.2	79.0	3.8	63.8	71.6	7.8	59.4	68.8	9.4
Coreutils	65.4	69.2	3.8	68.4	67.0	-1.4	59.8	65.6	5.8	60.6	65.2	4.6	48.8	55.0	6.2	51.0	57.2	6.2
ImageMagick	60.8	77.0	16.2	77.0	81.4	4.4	60.4	78.8	18.4	76.2	84.6	8.4	54.2	65.6	11.4	54.8	64.2	9.4
SQLite	74.8	88.4	13.6	82.6	85.8	3.2	74.6	86.0	11.4	88.4	86.4	-2.0	71.2	79.4	8.2	69.6	77.8	8.2
OpenSSL	81.8	90.8	9.0	76.2	84.2	8.0	85.2	85.0	-0.2	79.6	80.2	0.6	61.6	65.6	4.0	63.6	71.6	8.0
Putty	71.6	77.0	5.4	71.4	76.0	4.6	73.6	76.4	2.8	73.2	68.6	-4.6	61.0	69.0	8.0	59.2	64.6	5.4
Average	**73.0**	**82.5**	**9.5**	**77.1**	**80.7**	**3.5**	**72.7**	**79.4**	**6.7**	**76.3**	**78.1**	**1.8**	**61.3**	**68.3**	**7.0**	**60.5**	**67.9**	**7.5**

Table 4: Performance (MRR) improvement on the out-of-distribution dataset. Columns and rows can be interpreted similarly to Table 2.

Programs	$JTrans_{BC}$			$JTrans_{BK}$			$JTrans_{HS}$			$Trex_{BC}$			$Trex_{BK}$			$Trex_{HS}$		
	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.	Ori.	DiEmph	Impr.
Curl	55.3	71.9	16.6	63.9	65.9	2.0	60.8	62.4	1.6	60.0	61.5	1.5	47.1	50.7	3.6	38.4	46.0	7.6
Binutils	50.5	64.6	14.1	60.5	64.3	3.8	48.8	52.5	3.7	49.8	55.1	5.2	44.0	47.1	3.2	33.7	39.6	6.0
Coreutils	42.1	51.7	9.5	48.0	50.4	2.4	42.7	45.3	2.6	42.7	49.6	6.8	35.4	39.7	4.3	32.2	37.6	5.4
ImageMagick	34.0	51.8	17.8	53.6	57.5	3.9	41.1	54.6	13.5	51.7	57.7	6.0	36.3	44.1	7.9	32.9	41.1	8.2
SQLite	53.1	70.6	17.5	61.8	66.8	5.1	52.7	65.9	13.2	74.0	74.2	0.3	53.1	58.3	5.1	44.9	52.5	7.6
OpenSSL	58.4	64.8	6.4	55.9	62.0	6.0	65.5	69.2	3.7	57.3	60.3	3.0	42.3	47.7	5.4	40.0	45.2	5.2
Putty	46.8	59.2	12.4	47.7	49.4	1.7	52.5	55.3	2.8	50.3	48.1	-2.2	43.7	48.1	4.3	37.3	44.1	6.7
Average	**48.6**	**62.1**	**13.5**	**55.9**	**59.5**	**3.6**	**52.0**	**57.9**	**5.9**	**55.1**	**58.1**	**2.9**	**43.1**	**48.0**	**4.8**	**37.1**	**43.7**	**6.7**

Table 5: Effects of DiEmph on performance (PR@1) for the in-distribution dataset. The first column lists the binary programs. The model setups are listed in the first row in the form of $ModelArch_{dataset}$, where *BC* denotes BinaryCorp-3M, *BK* Binkit, and *HS* How-solve. In the second row, Ori. and DiEmph mean the PR@1 of the original model and the model improved by DiEmph, respectively. Diff. means the difference of performance between the two models.

Programs	$JTrans_{BC}$			$JTrans_{BK}$			$JTrans_{HS}$			$Trex_{BC}$			$Trex_{BK}$			$Trex_{HS}$		
	Ori.	DiEmph	Diff.	Ori.	DiEmph	Diff.	Ori.	DiEmph	Diff.	Ori.	DiEmph	Diff.	Ori.	DiEmph	Diff.	Ori.	DiEmph	Diff.
Curl	68.8	70.4	1.6	46.8	51.4	4.6	66.4	67.8	1.4	57.2	56.0	-1.2	27.0	36.0	9.0	43.4	43.8	0.4
Binutils	64.2	67.6	3.4	49.8	55.0	5.2	62.6	63.0	0.4	57.4	55.4	-2.0	26.2	36.2	10.0	36.8	43.6	6.8
Coreutils	34.6	44.0	9.4	31.6	44.6	13.0	36.4	40.4	4.0	38.0	38.2	0.2	15.2	24.8	9.6	20.0	25.4	5.4
ImageMagick	43.8	47.8	4.0	41.0	43.4	2.4	50.2	49.4	-0.8	44.8	43.6	-1.2	26.4	35.0	8.6	28.2	31.6	3.4
SQLite	66.6	70.2	3.6	40.6	46.2	5.6	58.0	55.8	-2.2	56.0	53.0	-3.0	26.0	30.4	4.4	39.2	41.8	2.6
OpenSSL	64.4	71.2	6.8	46.8	58.2	11.4	69.0	72.8	3.8	56.2	59.4	3.2	30.4	40.6	10.2	43.2	45.0	1.8
Putty	65.0	60.2	-4.8	51.4	50.2	-1.2	62.4	62.8	0.4	55.8	55.0	-0.8	24.4	35.8	11.4	42.8	47.0	4.2
Average	**58.2**	**61.6**	**3.4**	**44.0**	**49.9**	**5.9**	**57.9**	**58.9**	**1.0**	**52.2**	**51.5**	**-0.7**	**25.1**	**34.1**	**9.0**	**36.2**	**39.7**	**3.5**

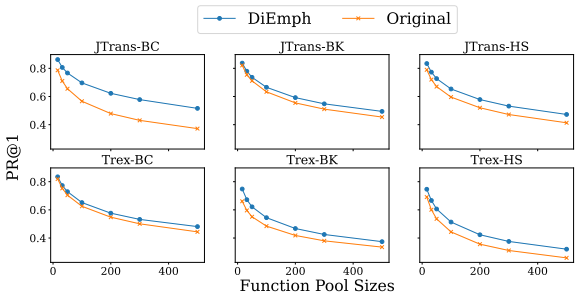

Figure 11: Effectiveness of DiEmph with different pool sizes. Each figure shows the performance of two models on a program in the out-of-distribution dataset. The x-axis denotes the sizes of candidate function pool. The y-axis denotes PR@1 (averaged over 7 programs in the testing dataset). The orange and blue lines denote the performance of the original model and the performance of the model improved by DiEmph, respectively.

Table 6: Run time efficiency of DiEmph (in minutes)

Model	KDE	Class. Imp.	All
$JTrans_{BC}$	4	9	15
$JTrans_{BK}$	10	22	33
$JTrans_{HS}$	13	25	39
$Trex_{BC}$	13	14	29
$Trex_{BK}$	14	16	29
$Trex_{HS}$	14	17	31
Average	**11**	**17**	**29**