TABLE: Overall performance of KILLBADCODE after filter by NCM.

Task	Code Poisoning	KillBadCode-Origin		KillBadCode-Filtered	
		FPR (%)	Recall (%)	FPR (%)	Recall (%)
Defect Detection	BadCode (Fixed)	3.81	100	3.56	100
	BadCode (Mixed)	5.18	100	4.84	100
	BNC (Fixed)	3.03	100	2.83	100
	BNC (Grammar)	14.88	100	13.89	100
	CodePoisoner (Variable)	23.43	100	17.19	100
	Average	10.07	100	8.46	100
Clone Detection	BadCode (Fixed)	2.50	100	2.33	100
	BadCode (Mixed)	11.98	100	8.45	100
	BNC (Fixed)	2.86	100	2.04	100
	BNC (Grammar)	12.39	100	9.26	100
	CodePoisoner (Variable)	15.58	100	11.04	100
	Average	9.06	100	6.62	100
Code Search	BadCode (Fixed)	1.11	100	0.96	100
	BadCode (Mixed)	1.38	100	1.25	100
	BNC (Fixed)	3.10	100	2.07	100
	BNC (Grammar)	4.69	100	3.62	100
	CodePoisoner (Variable)	20.31	100	13.57	100
	Average	6.12	100	4.29	100
Code Repair	BadCode (Fixed)	0.53	100	0.54	100
	BadCode (Mixed)	1.44	100	0.62	100
	BNC (Fixed)	1.53	100	0.51	100
	BNC (Grammar)	2.67	100	2.26	100
	CodePoisoner (Variable)	3.77	100	3.24	100
	Average	1.59	100	1.43	100

Current pre-training defenses all suffer from over-deletion (i.e., causing FPR), and KillBadCode is no exception. However, KillBadCode performs significantly better than baselines, achieving 100% recall while maintaining a low FPR. Additionally, our RQ2 results demonstrate that KillBadCode can maintain the overall model performance. To mitigate over-deletion, we propose a potentially feasible solution, called KillBadCode-Filtered. The dataset purified by KillBadCode can be used to train a clean NCM. The NCM can then be used to predict the labels of candidate poisoned samples, and ultimately, samples whose predicted labels differ from the original ones are removed. We applied this solution to four code tasks under five backdoor attacks, successfully reducing the FPR while maintaining 100% recall.