A ONLINE APPENDIX

Algorithm 2 GCG-based Trigger Inversion on Code Search Task

Input:	X^m	selected masked samples
	V	trigger vocabulary
	V_g	target vocabulary
	$f(\theta^*)$	backdoored NCM
	ϵ	times of iterations
	k	number of candidate substitutes
	r	times of repeat
	β	threshold for trigger anchoring
Output:	t^*	anchored trigger
	g^*	inverted target

1: **function** TriggerInversion(S^m, Q)
2: $t, g \leftarrow$ randomly initialize a trigger with n tokens and a target with m tokens from V and V_g, respectively
3: $\boldsymbol{e}_{S^m}, \boldsymbol{e}_Q \leftarrow$ produce embeddings of code snippets in S^m and embeddings of query in Q using $f(\theta^*)$
4: **for** $z = 0, z < \epsilon$, z++ **do**
5: $o_t, o_g \leftarrow$ generate the one-hot representation of t and the one-hot representation of g
6: $\boldsymbol{e}_t, \boldsymbol{e}_g \leftarrow$ produce o_t's embeddings and o_g's embeddings using $f(\theta^*)$
7: $\boldsymbol{e}'_{S^m} \leftarrow \boldsymbol{e}_{S^m} \oplus \boldsymbol{e}_t$
8: $\boldsymbol{e}'_Q \leftarrow \boldsymbol{e}_Q \oplus \boldsymbol{e}_g$
9: $G \leftarrow \nabla_{o_t} \mathcal{L}(f(\boldsymbol{e}'_{S^m}; \theta^*), \boldsymbol{e}'_Q)$
10: $G_g \leftarrow \nabla_{o_g} \mathcal{L}(f(\boldsymbol{e}'_{S^m}; \theta^*), \boldsymbol{e}'_Q)$
11: $\mathcal{T}, \mathcal{T}_g \leftarrow$ select substitutes for each trigger token based on top-k gradients of o_t in G and o_g in G_g, respectively
12: $t^C \leftarrow \emptyset$ ▷ store candidate substitute triggers
13: $g^C \leftarrow \emptyset$ ▷ store candidate substitute targets
14: **for** $j = 1, j < r, j++$ **do**
15: $t^j, g^j \leftarrow t, g$
16: $i, u \leftarrow$ randomly select a position to be replaced in t^j and g^j, respectively
17: $\mathcal{T}_i, \mathcal{T}_{gu} \leftarrow$ get all candidate substitutes for i-th token of t^j and u-th token of g^j, respectively
18: $t_i^j, g_u^j \leftarrow$ randomly select a substitute from $\mathcal{T}_i$ and $\mathcal{T}_{gu}$, respectively
19: $t^j, g^j \leftarrow$ replace the i-th token of t^j with t_i^j and the u-th token of g^j with g_u^j, respectively
20: $t^C \leftarrow t^C \cup t^j$
21: $g^C \leftarrow g^C \cup g^j$
22: **end for**
23: $x \leftarrow t^C \times g^C$ ▷ all possible ordered pairs of t^C and g^C
24: $t \leftarrow x_j.t, g \leftarrow x_j.g$, where $j = \arg\min_j \mathcal{L}(f(S^m \oplus x_j.t; \theta^*), Q \oplus x_j.g), j \in [1, r^2]$ ▷ compute best substitution
25: **end for**
26: **return** t, g
27: **end function**
28:
29: **function** TriggerAnchoring(S^m, Q, t, g)
30: $t^* \leftarrow \emptyset$
31: $l \leftarrow \mathcal{L}(f(S^m \oplus t; \theta^*), Q \oplus g)$
32: **for** each token t_i **in** t **do**
33: $l_i \leftarrow \mathcal{L}(f(S^m \oplus (t \setminus t_i); \theta^*), Q \oplus g)$
34: **if** $|l - l_i| > \beta$ **then**
35: $t^* \leftarrow t^* \cup t_i$
36: **end if**
37: **end for**
38: **return** t^*
39: **end function**
40:
41: $< S^m, Q > \leftarrow$ get masked code snippets and queries in X^m
42: $t, g^* \leftarrow$ TriggerInversion(S^m, Q)
43: $t^* \leftarrow$ TriggerAnchoring(S^m, Q, t, g^*)
44: **return** t^*, g^*

Algorithm 2 details the GCG-based trigger inversion of EliBadCode on the code search task. In addition to the selected masked samples (X^m), trigger vocabulary (V), a backdoored NCM ($f(\theta^*)$), times of iterations (ϵ), the number of candidate substitutes (k), times of repeat (r), and the threshold for trigger anchoring (β), EliBadCode also takes as input the target vocabulary V_g, which includes all possible tokens of the target. EliBadCode is necessary to simultaneously invert the target tokens when running GCG-based trigger inversion of EliBadCode on the code search task. Specifically, Algorithm 2 first gets masked code snippets (S^m) and the corresponding queries from (X^m) (line 41), then invokes the TriggerInversion function. In the TriggerInversion function, the processing of the trigger is the same as in Algorithm 1. Additionally, EliBadCode performs similar operations on the target. EliBadCode first randomly initializes a trigger (t) with n tokens and a target (g) with m tokens using V and V_g (line 2), respectively. Then EliBadCode transforms S^m and Q into vector representations (also called embeddings) $\boldsymbol{e}_{S^m}$ and $\boldsymbol{e}_Q$ using the embedding layer of $f(\theta^*)$ (line 3), respectively. Based on $\boldsymbol{e}_{S^m}$ and $\boldsymbol{e}_Q$, it further iteratively optimizes t and g ϵ times (lines 4–22), respectively. Notably, this process is similar to Algorithm 1, with the addition of optimization regarding g. Subsequently, it calculates the loss value l about the inverted trigger t and the inverted target g and returns them (lines 23–24). Next, the masked code sinppets S^m, queries$Q$, inverted trigger t and invert target g^* will be input into the TriggerAnchoring function to obtain the effective components of the inverted trigger t^*. Finally, Algorithm 2 returns the anchored trigger t^* and inverted target g^*.