UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Precise and Interpretable Editing of Code Knowledge in Large Language Models

Min Xue, Nikolai Bolik, Lennart Stöpler, Erik Imgrund, Janik Schmid, Artur Andrzejak

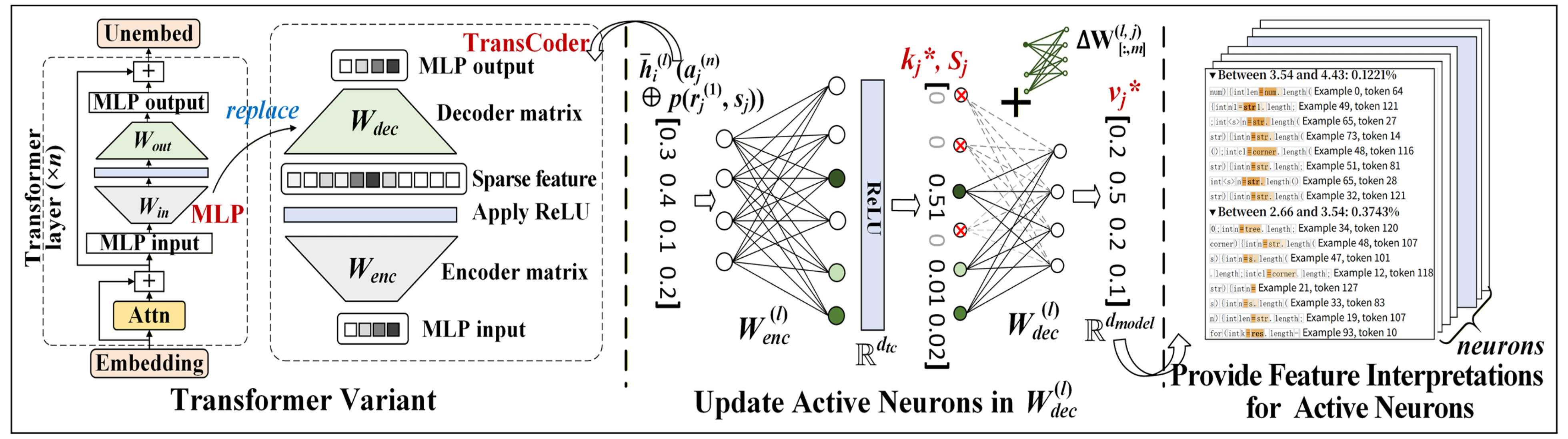
ICLR
2026

Overview

- **Motivation:**
  - Knowledge Editing (KE) offers a lightweight alternative to computationally expensive fine-tuning for updating LLMs.
  - Existing KE methods are limited by the polysemanticity of MLP neurons, making it difficult to understand how knowledge is injected into weights.
- **Our Main Contributions:**
  - **TransCoder-based Precise Editing (TCPE)**: Leveraging the sparsity and monosemanticity of TransCoder neurons for precise editing.
  - **KECode**: A benchmark for evaluating code-to-code translation based on functional equivalence.

Approach

TCPE: Neuron-Level Sparse Update Mechanism



- We replace the original MLP with TransCoder, an MLP-like model component with a wide and sparsely activated hidden feature vector.
- Consider T new key-value pairs

$$\{(k_j^*, v_j^*, S_j)\}_{j=1}^T, \quad k_j^* \in \mathbb{R}^{d_{\text{tc}}}, \; v_j^* \in \mathbb{R}^{d_{\text{model}}} .$$

where each (k_j^*, v_j^*) encodes the j-th correction knowledge $(r_j^{(1)}, s_j, r_j^{(2)}, o_j^*)$. the set S_j contains the indices of the active neurons in k_j^*.

- Following ROME, we compute the update matrix $\Delta \mathbf{W}^{(l,j)}$ and apply sparse updates restricted to the active neurons S_j.

$$\mathbf{W}_{\text{dec}}^{(l)'}[:, m] = \mathbf{W}_{\text{dec}}^{(l)}[:, m] + \Delta \mathbf{W}^{(l,j)}[:, m], \quad \forall m \in S_j \ , \qquad (1)$$

KECode: Knowledge Editing Benchmark in Code Translation

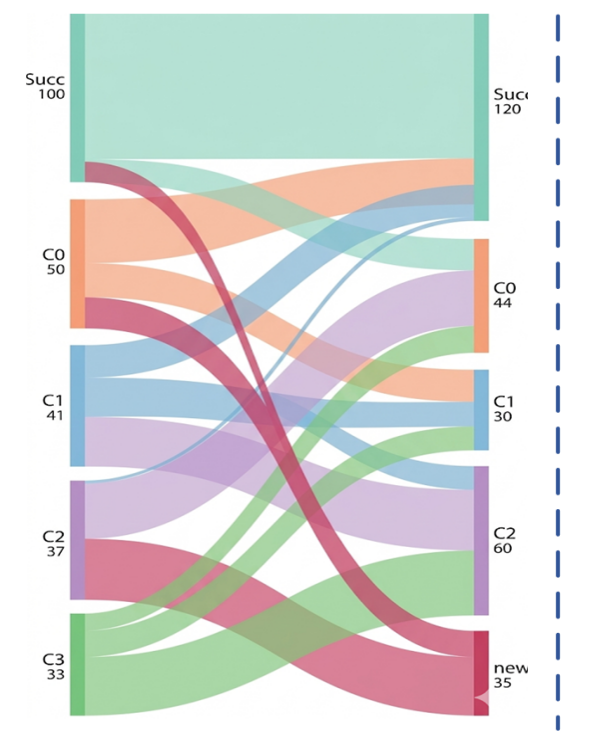
- **Why KECode?** Code requires strict syntactic and semantic correctness.
- **Dataset:** Java→D translation, 600 samples with unit tests.
- **Evaluation Metrics:**

Generalization: corrected target errors

Cluster Drift: change in target error cluster

Specificity: impact on non-target error cluster

Reliability: stability across test cases



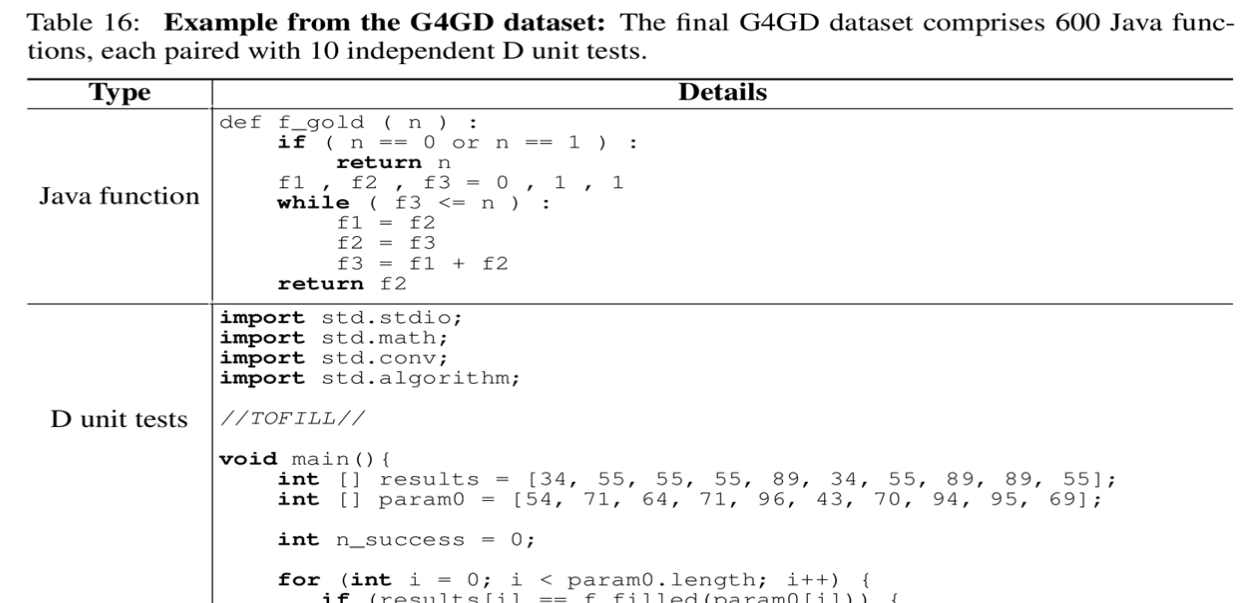


Table 16: **Example from the G4GD dataset:** The final G4GD dataset comprises 600 Java functions, each paired with 10 independent D unit tests.

Type	Details
Java function	

```
def f_gold ( n ) :
    if ( n == 0 or n == 1 ) :
        return n
    f1 , f2 , f3 = 0 , 1 , 1
    while ( f3 <= n ) :
        f1 = f2
        f2 = f3
        f3 = f1 + f2
    return f2
```

Type	Details
D unit tests	

```
import std.stdio;
import std.math;
import std.conv;
import std.algorithm;

//TOFILL//

void main(){
    int [] results = [34, 55, 55, 55, 89, 34, 55, 89, 89, 55];
    int [] param0 = [54, 71, 64, 71, 96, 43, 70, 94, 95, 69];

    int n_success = 0;

    for (int i = 0; i < param0.length; i++) {
      if (results[i] == f_filled(param0[i])) {
                n_success += 1;
            }}

    writefln("#Results:%d,%d", n_success, param0.length);
}
```

Technical Details

Code Error Clustering & Correction Knowledge Construction

- Identify the main clusters of code errors.
- Construct correction knowledge for code translation tasks.

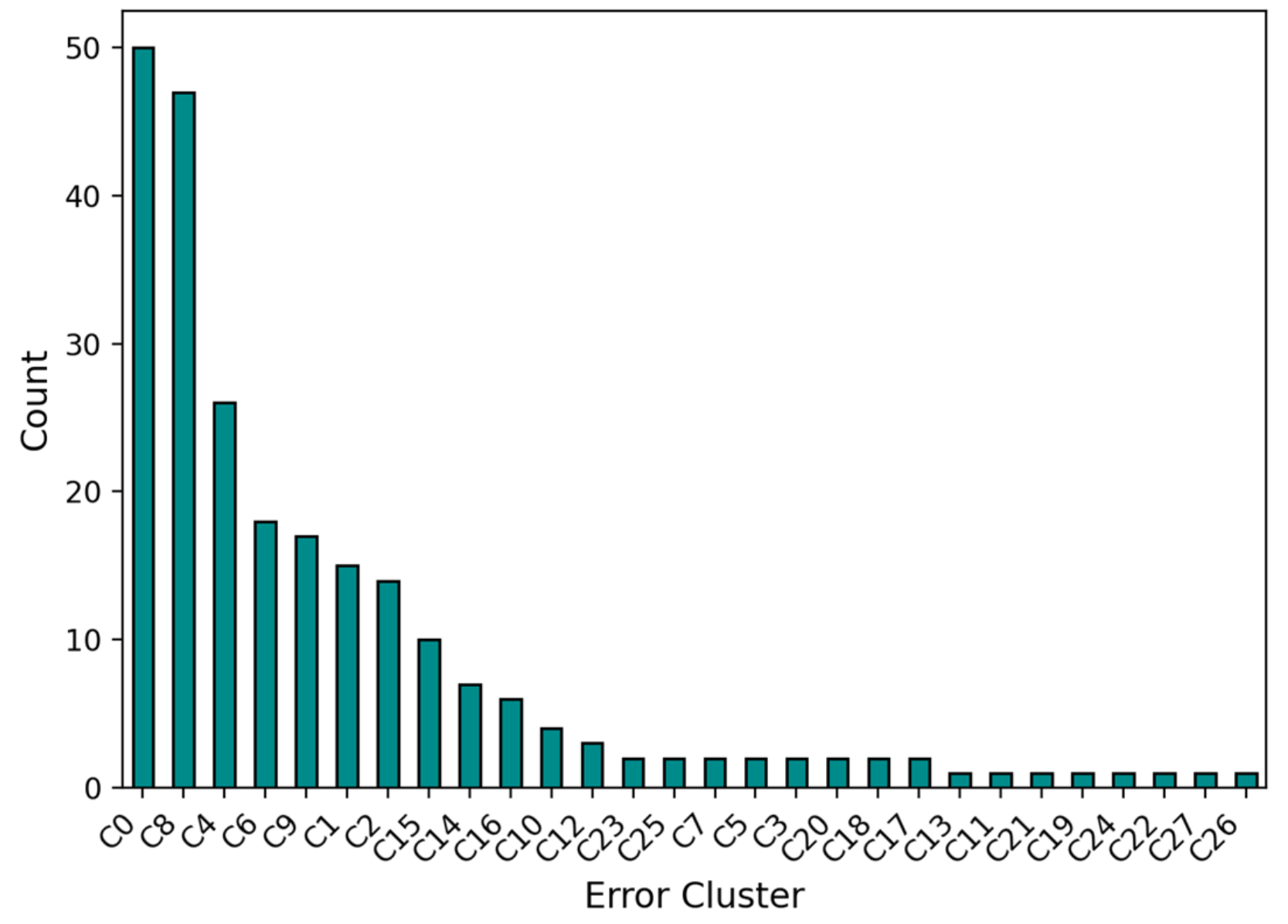
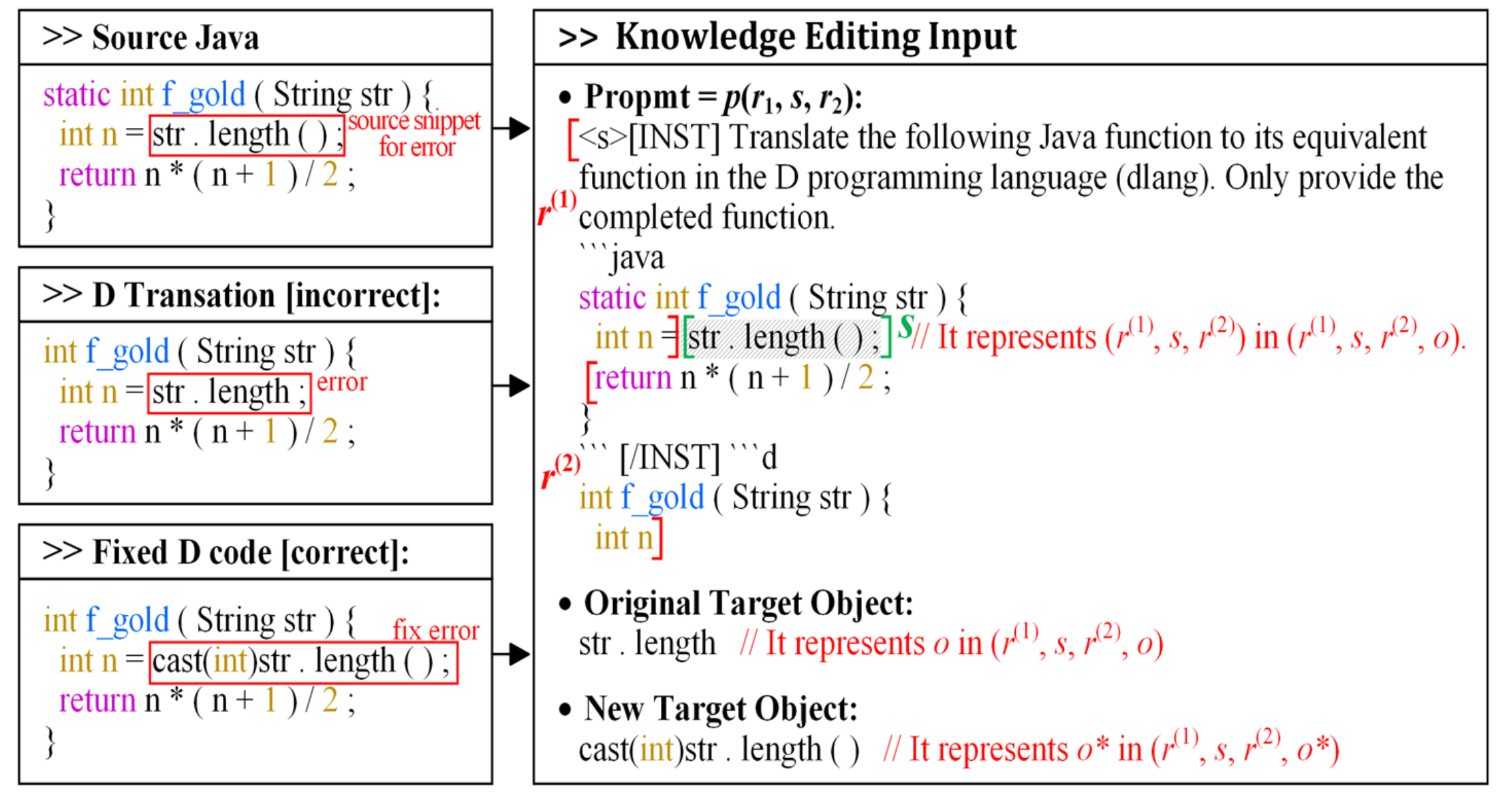


Figure 2. Error Clusters of CodeLlama on Java-to-D Translation Tasks. The top four clusters, C_0, C_8, C_4, and C_6, account for 58.5% of all compilation failures.



>> Source Java

```
static int f_gold ( String str ) {
  int n = str . length ( ) ;
  return n * ( n + 1 ) / 2 ;
}
```

>> D Transation [incorrect]:

```
int f_gold ( String str ) {
  int n = str . length ;
  return n * ( n + 1 ) / 2 ;
}
```

>> Fixed D code [correct]:

```
int f_gold ( String str ) {
  int n = cast(int)str . length ( ) ;
  return n * ( n + 1 ) / 2 ;
}
```

>> Knowledge Editing Input

- **Propmt = $p(r_1, s, r_2)$:**

~~~
<s>[INST] Translate the following Java function to its equivalent
function in the D programming language (dlang). Only provide the
completed function.
```java
static int f_gold ( String str ) {
  int n = str . length ( ) ; // It represents (r(1), s, r(2)) in (r(1), s, r(2), o).
  return n * ( n + 1 ) / 2 ;
}
``` [/INST] ```d
int f_gold ( String str ) {
  int n
~~~

- **Original Target Object:**
str . length // It represents o in $(r^{(1)}, s, r^{(2)}, o)$
- **New Target Object:**
cast(int)str . length ( ) // It represents o^* in $(r^{(1)}, s, r^{(2)}, o^*)$

Figure 3. Construction of the Correction Knowledge Four-Tuple ($r^{(1)}$, s, $r^{(2)}$, o*) for Java-to-D code translation, targeting the C_4 error cluster: *C-style cast illegal, use 'cast(int)x'.*

Analysis of Active Neuron Overlap Across and Within Clusters

Let $S_{i,j}$ be the activated neuron indices for prompt j in cluster C_i.

- **Average size:** A_1 (cross-cluster), A_2 (intra-cluster).
- **Union:** U_1 (cross-cluster), U_2 (intra-cluster).
- **Intersection:** I_2 (intra-cluster).
- **Absolute Overlap (AO):** $\text{AO} = \frac{|\bigcup_{i=1}^{N}\bigcup_{j=1}^{k} S_{i,j}|}{\sum_{i=1}^{N}|\bigcup_{j=1}^{k} S_{i,j}|}$
- **Relative Overlap (RO):** $\text{RO} = \frac{|\bigcup_{i=1}^{N}\bigcup_{j=1}^{k} S_{i,j}|}{\frac{1}{Nk}\sum_{i=1}^{N}\sum_{j=1}^{k}|S_{i,j}|}$

Table 10: **Comparison of Cross-Cluster Active Neuron Overlap**

layer	A_1	U_1	RO	AO
MLP	5514	10836	1.97	0.16
LTC4	117	848	7.25	0.44
LTC8	132	1059	8.02	0.47
LTC16	448	2987	6.67	0.41

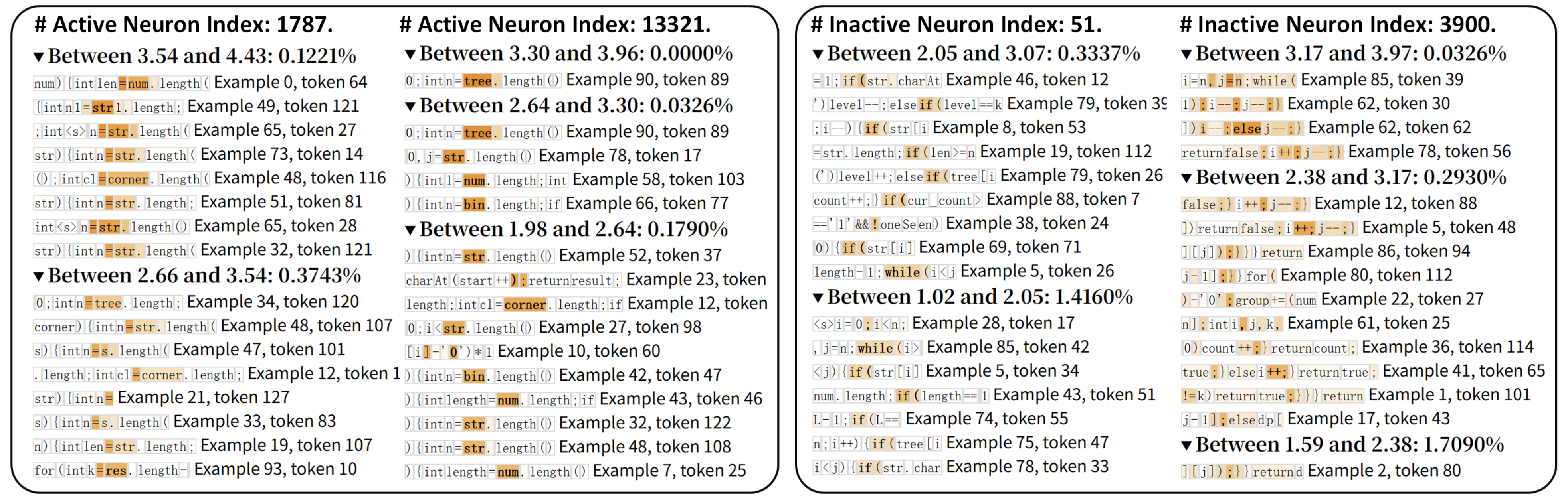
Table 11: **Intra-Cluster Active Neuron Counts and Overlaps**

	Stats	C_0	C_1	C_2	C_4	C_6	C_8	C_9	C_{15}	AVE
CodeLlama	A_2	6455	7502	9144	9279	9305	7548	8214	7413	8108
	I_2	4543	3538	1841	1822	1715	3449	2807	3646	2920
	U_2	5495	5520	5470	5521	5500	5532	5533	5538	5514
LTC4	A_2	138	111	108	116	124	113	109	115	117
	I_2	95	51	20	20	25	51	34	64	45
	U_2	191	185	270	306	339	200	236	202	241
LTC8	A_2	147	131	127	134	138	126	122	131	132
	I_2	91	67	25	29	29	61	43	63	51
	U_2	223	214	328	369	389	232	284	233	284
LTC16	A_2	551	452	427	440	430	424	396	454	447
	I_2	386	236	73	92	76	198	123	263	181
	U_2	759	736	1068	1172	1208	746	885	754	916

Experiments

Neuron-Level Interpretability in Knowledge Editing

- Focusing on typical D-type conversion issues like "*Error: cannot implicitly convert expression 'str.length' of type 'ulong' to 'int'*".



# Active Neuron Index: 1787.

▾Between 3.54 and 4.43: 0.1221%
num){intlen=num.length(Example 0, token 64
{intn1=str1.length; Example 49, token 121
;int<s>n=str.length(Example 65, token 27
str){intn=str.length(Example 73, token 14
();intcl=corner.length(Example 48, token 116
str){intn=str.length; Example 51, token 81
int<s>n=str.length() Example 65, token 28
str){intn=str.length(Example 32, token 121
▾Between 2.66 and 3.54: 0.3743%
0;intn=tree.length; Example 34, token 120
corner){intn=str.length(Example 48, token 107
s){intn=s.length(Example 47, token 101
.length;intcl=corner.length; Example 12, token 1
str){intn= Example 21, token 127
s){intn=s.length(Example 33, token 83
n){intlen=str.length; Example 19, token 107
for(intk=res.length- Example 93, token 10

# Active Neuron Index: 13321.

▾Between 3.30 and 3.96: 0.0000%
0;intn=tree.length() Example 90, token 89
▾Between 2.64 and 3.30: 0.0326%
0;intn=tree.length() Example 90, token 89
0,j=str.length() Example 78, token 17
){intl=num.length;int Example 58, token 103
){intn=bin.length;if Example 66, token 77
▾Between 1.98 and 2.64: 0.1790%
){intn=str.length() Example 52, token 37
charAt(start++);returnresult; Example 23, token
length;intcl=corner.length;if Example 12, token
0;i<str.length() Example 27, token 98
[i]-'0')*1 Example 10, token 60
){intn=bin.length() Example 42, token 47
){intlength=num.length;if Example 43, token 46
){intn=str.length() Example 32, token 122
){intn=str.length() Example 48, token 108
){intlength=num.length() Example 7, token 25

# Inactive Neuron Index: 51.

▾Between 2.05 and 3.07: 0.3337%
=1;if(str.charAt Example 46, token 12
')level--;elseif(level==k Example 79, token 39
;i--){if(str[i Example 8, token 53
=str.length;if(len>=n Example 19, token 112
(')level++;elseif(tree[i Example 79, token 26
count++;}if(cur_count> Example 88, token 7
=='1'&&!oneSeen) Example 38, token 24
0){if(str[i] Example 69, token 71
length-1;while(i<j Example 5, token 26
▾Between 1.02 and 2.05: 1.4160%
<s>i=0;i<n; Example 28, token 17
,j=n;while(i> Example 85, token 42
<j){if(str[i] Example 5, token 34
num.length;if(length==1 Example 43, token 51
L-1;if(L== Example 74, token 55
n;i++){if(tree[i Example 75, token 47
i<j){if(str.char Example 78, token 33

# Inactive Neuron Index: 3900.

▾Between 3.17 and 3.97: 0.0326%
i=n,j=n;while(Example 85, token 39
1);i--;j--;} Example 62, token 30
])i--;elsej--;} Example 62, token 62
returnfalse;i++;j--;} Example 78, token 56
▾Between 2.38 and 3.17: 0.2930%
false;}i++;j--;} Example 12, token 88
])returnfalse;i++;j--;} Example 5, token 48
][j]);}}}return Example 86, token 94
j-1];}}for(Example 80, token 112
)-'0';group+=(num Example 22, token 27
n];inti,j,k, Example 61, token 25
0)count++;}returncount; Example 36, token 114
true;}elsei++;}returntrue; Example 41, token 65
!=k)returntrue;}}}return Example 1, token 101
j-1];elsedp[Example 17, token 43
▾Between 1.59 and 2.38: 1.7090%
][j]);}}returnd Example 2, token 80

Analysis of TCPE and Baselines in a Multi-Error Editing Scenario

Table 2: **Comparative Performance of Knowledge Editing Methods in a Multi-Error Editing Scenario.** *Score* quantifies overall knowledge editing performance and is calculated as Score = (GN − CD) + (LoC − DT) + RE.

Method	Score↑	Efficacy		Specificity		Reliability		
		GN↑	CD ↓	LoC↑	DT ↓	RE↑	AC_{post}	AC_{pre}
FT	86.40	7.09	100.00	87.36	6.38	97.97	56.33	57.50
Few-shot	120.20	12.06	87.23	90.20	2.61	107.78	61.33	57.50
WISE	99.74	3.55	98.58	96.51	1.74	100.00	57.50	57.50
AGRACE	99.85	0.71	100.71	99.56	0.00	100.29	57.67	57.50
FiNE	144.50	29.08	54.61	81.92	13.62	101.74	58.50	57.50
LoRA	147.27	29.08	58.87	82.57	10.44	104.93	60.33	57.50
ROME	109.60	29.79	80.14	76.47	16.23	99.71	57.33	57.50
MEMIT	14.96	7.09	124.82	72.99	22.32	82.03	47.17	57.50
PMET	-4.70	12.77	104.26	55.77	38.55	69.57	40.00	57.50
TCPE (LTC_{mlp})	171.59	30.61	**44.90**	83.00	6.05	108.93	63.00	57.83
TCPE (LTC4)	171.45	**32.86**	46.43	82.17	**6.86**	109.71	**64.00**	58.33
TCPE (LTC8)	**174.82**	31.66	49.64	**88.50**	5.73	**110.03**	**64.00**	58.17

Table 15: **Example Prompt and Corresponding Output: Translating a Java Function to the D Language.**

Type	Details
Prompt & Output	

~~~
<s> [INST] Translate the following Java function to its equivalent
    function in the D programming language (dlang). Only provide the
    completed function.
```int solution (String s) {
    int result = 0;
    int n = s.length ();
    for (int i = 0;  i < n;  i++)
        for (int j = i;  j < n;  j++)
            if (s.charAt (i) == s.charAt (j))
                result++;
    return result;
}
``` [/INST] ```d
int solution(string s) {
    int result = 0;
    int n = cast(int) s.length;
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            if (s[i] == s[j])
                result++;
        }
    }
    return result;
}
``` </s>
~~~

Information-Carrying Role of Active Neurons for Precise Editing

Table 3: **Role of Active TransCoder Neurons in Multi-Error Editing.** Neurons are updated if activations acv exceed thresholds $\tau \in \{0, 0.001, 0.01, 0.05, 0.08, 0.1, 0.15, 0.2\}$. "∪" and "∩" denote the union and intersection of updated neurons across multiple error types.

Method	Efficacy		Specificity		Reliability			Neurons	
	GN↑	CD↓	LoC↑	DT↓	RE↑	AC_{post}	AC_{pre}	∪	∩
LTC4									
acv>0.2	24.29	59.29	82.39	7.43	105.43	61.50	58.33	34	3
acv>0.1	32.86	46.43	82.17	6.86	109.71	64.00	58.33	69	3
acv>0.08	34.29	52.86	81.96	9.43	106.57	62.17	58.33	78	4
acv>0.05	35.00	48.57	82.39	8.57	106.86	62.33	58.33	98	4
acv>0.01	33.57	50.00	83.04	8.29	106.29	62.00	58.33	135	4
acv>0.001	32.86	50.71	83.04	8.29	106.00	61.83	58.33	147	5
acv>0	32.86	50.71	83.04	8.29	106.00	61.83	58.33	147	5
LTC8									
acv>0.2	29.50	51.80	88.29	6.59	108.31	63.00	58.17	39	1
acv>0.1	32.37	46.76	87.42	6.59	109.74	63.83	58.17	70	3
acv>0.08	31.66	49.64	88.50	5.73	110.03	64.00	58.17	92	3
acv>0.01	31.66	51.08	82.65	10.32	105.73	61.50	58.17	181	5
acv>0.001	31.66	51.08	82.65	10.32	105.73	61.50	58.17	195	5
acv>0	31.66	51.08	82.65	10.32	105.73	61.50	58.17	197	5

Table 4: **Ablation on Active TransCoder Neurons in Multi-Error Editing.** Neurons are updated if activations acv below thresholds $\tau \in \{0, 0.001, 0.01, 0.05, 0.08, 0.1\}$. Here, "Updated Neurons" denotes the number of neurons that meet each threshold.

Method	Efficacy		Specificity		Reliability			Updated Neurons
	GN↑	CD↓	LoC↑	DT↓	RE↑	AC_{post}	AC_{pre}	
LTC4								
acv =0	1.43	101.43	98.48	0.57	100.29	58.5	58.33	16,237
acv ≤0.001	1.43	101.43	98.48	0.57	100.29	58.5	58.33	16,237
acv ≤0.01	1.43	101.43	98.26	0.57	100.29	58.5	58.33	16,249
acv ≤0.05	1.43	101.43	98.04	0.86	100	58.33	58.33	16,286
acv ≤0.08	1.43	101.43	98.04	0.86	100	58.33	58.33	16,306
acv ≤0.1	7.14	92.86	96.09	1.71	102	59.5	58.33	16,315
LTC8								
acv =0	1.44	100	99.13	0.29	100.57	58.5	58.17	32,571
acv ≤0.001	1.44	100	99.13	0.29	100.57	58.5	58.17	32,573
acv ≤0.01	1.44	100	99.13	0.29	100.57	58.5	58.17	32,587
acv ≤0.05	1.44	100	99.13	0.29	100.57	58.5	58.17	32,646
acv ≤0.08	1.44	99.28	99.13	0.29	100.86	58.67	58.17	32,676
acv ≤0.1	1.44	99.28	98.92	0.57	100.57	58.5	58.17	32,698

https://aip.ifi.uni-heidelberg.de/publications

ICLR Conference 2026, Rio de Janeiro, Brazil

min.xue@uni-heidelberg.de