International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(23s) 2026 pp. 259-273
https://doi.org/10.70917/ijcisim-2026-5572
Article

Graph Neural Networks with Code Structure Embeddings for Model-Agnostic Multi-Class Detection of AI-Generated Programs

Dr. Hayder Kareem Algabri

Department of Engineering Cybersecurity Techniques , College of Engineering Techniques,University of Hilla.
hayder_kareem_algabri@hilla-unc.edu.iq

Abstract: Most detectors of AI-generated code answer a binary question: human or machine. We study the multi-class problem, naming the generator family behind a program, and the open-set problem that follows from it, in which the generator was never seen during training. Our detector, GraphCSE, represents a program as a heterogeneous code graph with four edge relations and fuses structural (syntax-type) and semantic (token) node channels through a learned per-node gate before relation-aware graph attention. On a controlled synthetic corpus of 2,500 Python programs spanning a human class built from eight style archetypes and four generator-family proxies, GraphCSE reaches 92.1 ± 0.6% macro-F1, the strongest learned model we test and 11.2 points above a GCN on identical graphs, though character n-gram TF-IDF remains slightly ahead in-distribution (93.4 ± 0.5%). The paper's central findings concern unseen generators. First, a negative one: on single programs, all five rejection rules we test (closed-set argmax, centroid and k-NN distance, energy, maximum softmax probability) fail on held-out families, with mean AUROC no better than 0.65: a tight unseen style inside the human envelope is indistinguishable from an unusual human. Second, a constructive one: judging a set of programs from the same source by a two-sided anomaly test on embedding statistics, calibrated only on human data, restores model-agnostic detection, reaching 86.4-97.3% mean detection at ten programs per source (AUROC up to 0.972) in the graph embeddings versus 53.6% in raw TF-IDF space. Provenance of unseen generators is a property of collections, not snippets.

Keywords: AI-generated code detection; graph neural networks; code provenance; heterogeneous attention; open-set recognition; abstract syntax trees

1. Introduction

A growing share of the code that reaches version control was never typed by a person. Code-generating models such as Codex [1], Code Llama [2], and StarCoder [3] now sit inside mainstream development workflows, and instructors report that distinguishing student work from model output has become a routine assessment problem [4]. Provenance matters beyond the classroom: security teams triage machine-written patches differently from human ones, license compliance depends on where code came from, and incident forensics increasingly asks not just whether a model wrote a snippet but which model did.

Most published detectors answer a simpler question. GPTSniffer [5] and related classifiers frame detection as a binary decision between human-written and ChatGPT-written code, and zero-shot approaches adapt text detectors such as DetectGPT to the code domain [6], [7]. Watermarking offers provenance by construction [8], but only for cooperating providers, and theoretical results caution that post-hoc text detection degrades as generators improve [9]. Recent stylometric work shows that individual language models leave measurable fingerprints in the code they emit, enough to attribute C programs to one of several commercial models with high accuracy [10]. What these attribution results leave open is robustness: a classifier trained to separate a fixed roster of generators has no obvious reason to behave sensibly when a new model, or a new fine-tune of an old one, enters circulation.

We approach the problem through structure. Two decades of code-authorship research showed that syntactic features survive superficial edits better than lexical ones [11], [12], and graph representations of programs have since become the standard way to expose that structure to neural models [13], [14]. Our hypothesis is that generator families differ both in how they shape a program (control-flow habits, factoring, guard density) and in what they name things (identifier conventions, docstring phrasing), that the informative balance between the two varies from node to node, and that a detector should therefore learn that balance rather than fix it. We instantiate this idea as GraphCSE, a graph neural network over a four-relation code graph whose input layer fuses a structural and a semantic channel through a per-node sigmoid gate, and whose message passing applies relation-specific attention in the spirit of heterogeneous graph models [17], [18].

A second concern shapes the experimental design. Evaluating multi-class provenance on real model outputs conflates two sources of difficulty, the detector and the corpus, and gives the experimenter no control over how far apart the classes actually are. We instead build a fully synthetic corpus of parameterized Python program generators: a human class drawn from a wide distribution of idiosyncratic styles and four generator-family proxies with distinct, low-variance stylistic centroids, plus a fifth family reserved as a truly unseen test population. Synthetic data cannot settle how well GraphCSE detects any specific commercial model, and we claim nothing of the sort; what it buys is a controlled testbed in which class geometry is known, experiments are exactly reproducible on a CPU, and the open-set question can be asked cleanly.

The contributions of this paper are as follows.

(1) We formulate model-agnostic detection of AI-generated programs as joint multi-class attribution and open-set rejection, on a controlled synthetic benchmark of 2,800 syntactically valid Python programs whose human class is an eight-archetype style mixture and whose five generator families carry family-specific phrasing registers, one family deliberately overlapping a human archetype and one reserved as fully unseen.

(2) We propose GraphCSE, a four-relation heterogeneous code graph encoder with per-node gated fusion of structural and semantic channels (HSSA) and relation-aware attention; it is the strongest learned model in-distribution at 92.1 ± 0.6% macro-F1, and our ablations honestly locate its margin in the dual channels rather than in the learned gate, which does not beat fixed uniform fusion.

(3) We establish a robust negative result: on single programs, five standard open-set decision rules all fail to detect held-out generator families (closed-set detection 19.9% on average for GraphCSE; the best score-based rule averages 0.65 AUROC), a failure we replicate across scoring paradigms and trace to unseen styles falling inside the human stylistic envelope.

(4) We show that a set-level two-sided anomaly test on two embedding statistics, mean k-NN distance to the human training bank and within-set dispersion, calibrated on human data alone, recovers detection as set size grows: at ten programs per source the graph embeddings reach 86.4-97.3% mean detection across five unseen families, far above raw TF-IDF space (53.6%).

The remainder of the paper is organized in the usual way: Section 2 positions the work, Section 3 details the method, Sections 4 and 5 describe the experimental setup and results, and Section 6 concludes.

2. Related Work

2.1 Detecting machine-generated text and code

Detection of machine-generated natural language predates code-specific work. Supervised classifiers over contextual embeddings [27], visual-statistical tools such as GLTR [28], and curvature-based zero-shot tests such as DetectGPT [7] established the main families of techniques, while Sadasivan et al. [9] argued that detectability shrinks as generator and human distributions converge. Code turned out to be a harder target than prose: its low token entropy and rigid syntax break assumptions that text detectors rely on. Yang et al. [6] found existing detectors close to chance on code and modified DetectGPT with a surrogate scoring model; perplexity-based screening of student submissions

has been studied directly [24], with follow-up work questioning how far perplexity alone can carry detection [26]. Ye et al. [25] exploit the observation that a model rewrites its own code with little variation. On the supervised side, GPTSniffer fine-tunes CodeBERT for the binary task [5]. Nearly all of this line treats detection as two-way. Our setting differs in both directions at once: we ask for the generator family, not just the fact of generation, and we measure what happens when the family was never in the training roster.

2.2 Code authorship and model attribution

Attributing code to human authors is a mature problem: Caliskan-Islam et al. [11] de-anonymized programmers from syntax-tree features, and Abuhamad et al. [12] scaled deep authorship identification to thousands of authors. The same question is now being asked of models. Uchendu et al. [29] framed multi-way attribution among text generators; for code, Bisztray et al. [10] attribute C programs among five commercial LLMs with fine-tuned transformers, and report that comment phrasing is the single strongest cue. These systems assume the candidate list is closed. The open-set gap they leave, a new generator appearing after training, is precisely the failure mode our Section 5.3 quantifies, and the reason we treat embedding-space rejection as part of the detector rather than an afterthought, following the out-of-distribution scoring tradition of Lee et al. [30].

2.3 Graph representations of programs

Representing programs as graphs rather than token sequences goes back at least to Allamanis et al. [13], who augmented syntax trees with data-flow edges for variable-misuse tasks, and to path-based embeddings such as code2vec [14]. Graph neural networks over such structures, from GGNN [23] to convolutional [15] and attention-based [16] formulations, have served vulnerability detection [22] and pre-trained code models that inject data flow into transformers [20], complementing token-only pre-training [19], [21]. Heterogeneous attention, weighting neighbors differently per relation type, was developed for typed information networks [17] and relational knowledge graphs [18]. Unlike these works, which target semantics-centered tasks, we use the graph machinery for provenance, where the signal is stylistic, and we make the structural-versus-semantic balance itself a learned, per-node quantity, which none of the cited models does.

3. Proposed Method

GraphCSE takes a source file, builds a heterogeneous graph from its syntax tree, fuses two node-level feature channels through a learned gate, propagates information with relation-aware attention, and classifies a pooled graph embedding. Figure 1 shows the pipeline; the subsections follow the order in which data flows through it. Throughout, bold lower-case symbols denote vectors, bold upper-case symbols matrices, and ‖ denotes concatenation.

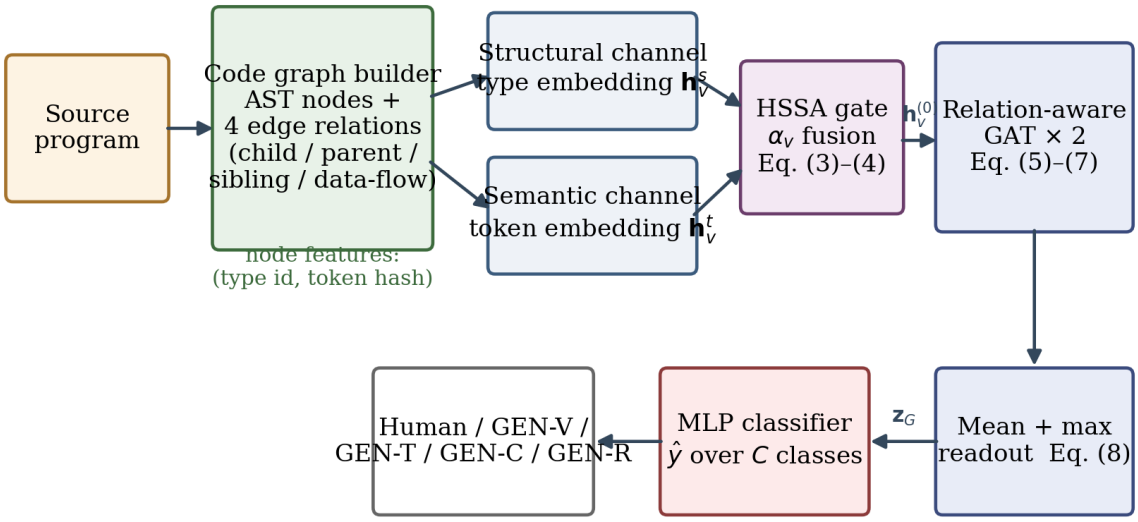

Figure 1. Overall architecture of GraphCSE. A program is parsed into a four-relation code graph; each node carries a structural (syntax-type) and a semantic (token) embedding, fused per node by the HSSA gate before two relation-aware graph attention layers and a mean-max readout.

3.1 Heterogeneous code graph construction

A program is parsed into its abstract syntax tree (AST), and every AST node becomes a graph node. Four directed edge relations connect them, as defined in Eq. (1): syntactic parent-to-child edges (chd), their reverses (par), next-sibling edges between consecutive children of the same parent (sib), which encode statement order, and def-use data-flow edges (dfg) linking a variable's assignment to its subsequent reads within the enclosing function scope, in the spirit of [13].

$$G = (\mathcal{V},\ \{\mathcal{E}_r\}_{r\in\mathcal{R}}), \qquad \mathcal{R} = \{\text{chd},\ \text{par},\ \text{sib},\ \text{dfg}\} \tag{1}$$

Each node v carries two raw attributes: its syntax type τ(v), one of the parser's node categories (For, Call, BinOp, and so on), and its lexical token σ(v), the identifier, attribute, or literal attached to the node when one exists. Tokens are mapped into 512 buckets by a stable hash so the vocabulary stays bounded; nodes without a token share a null bucket. Graphs are truncated at 400 nodes, a limit that no program in our corpus reaches.

3.2 Structural and semantic channels with HSSA fusion

Two embedding tables turn the raw attributes into the structural channel and the semantic channel of Eq. (2).

$$\mathbf{h}_v^s = \mathbf{E}_s[\tau(v)], \qquad \mathbf{h}_v^t = \mathbf{E}_t[\sigma(v)] \tag{2}$$

The channels answer different questions. The structural vector says what kind of syntactic object the node is, which captures how a generator shapes programs; the semantic vector says what the node is called, which captures naming and phrasing habits. For a docstring literal the words are nearly all the evidence; for an arithmetic operator the type is. Because the informative channel varies node by node, we let the network decide per node: a small gating network produces a scalar α_v from both channels, Eq. (3), and the input representation is the α-weighted convex combination of the two projected channels, Eq. (4). We call this mechanism Heterogeneous Structural-Semantic Attention (HSSA).

$$\alpha_v = \mathrm{sigmoid}\big(\mathbf{w}_g^{\top} \tanh\big(\mathbf{W}_g\,[\mathbf{h}_v^s \,\|\, \mathbf{h}_v^t] + \mathbf{b}_g\big)\big) \tag{3}$$

$$\mathbf{h}_v^{(0)} = \alpha_v\,\mathbf{W}_s\mathbf{h}_v^s + (1-\alpha_v)\,\mathbf{W}_t\mathbf{h}_v^t \tag{4}$$

Fixing α_v = 0.5 recovers plain additive fusion, an ablation evaluated in Section 5.2; the learned gate values are examined directly in Section 5.2, where they turn out to tell an honest and less flattering story than this design argument anticipates.

3.3 Relation-aware graph attention

Message passing must respect the fact that a data-flow neighbor and a sibling carry different kinds of evidence. Each layer therefore maintains, for every relation r, its own projection W_r and additive attention parameters, Eq. (5), normalizes attention over each node's incoming r-neighborhood, Eq. (6), and combines the per-relation aggregates through a softmax-normalized relation weight vector γ, Eq. (7), together with a self-loop projection, an ELU nonlinearity, and layer normalization. The design follows the additive attention of GAT [16] and the per-relation parameterization of R-GCN [18]; the learned γ plays the role of the semantic-level attention in HAN [17], collapsed to a single vector since our relation set is small.

$$e_{uv}^r = \mathrm{LeakyReLU}\big(\mathbf{a}_{r,\mathrm{src}}^{\top}\mathbf{W}_r\mathbf{h}_u + \mathbf{a}_{r,\mathrm{dst}}^{\top}\mathbf{W}_r\mathbf{h}_v\big) \tag{5}$$

$$\beta_{uv}^r = \frac{\exp(e_{uv}^r)}{\sum_{u' \in \mathcal{N}_r(v)} \exp(e_{u'v}^r)} \tag{6}$$

$$\mathbf{h}_v^{(l+1)} = \mathrm{LN}\Big(\mathrm{ELU}\Big(\mathbf{W}_o\mathbf{h}_v^{(l)} + \sum_{r\in\mathcal{R}}\gamma_r \sum_{u\in\mathcal{N}_r(v)} \beta_{uv}^r\,\mathbf{W}_r\mathbf{h}_u^{(l)}\Big)\Big), \quad \boldsymbol{\gamma} = \mathrm{softmax}(\mathbf{g}) \tag{7}$$

3.4 Readout, training objective, and complexity

After L layers, node states are pooled by concatenated mean and max readout, Eq. (8), and a two-layer MLP head maps the graph embedding z_G to class logits. Training minimizes cross-entropy with weight decay, Eq. (9). Algorithm 1 summarizes one forward pass.

$$\mathbf{z}_G = \Big[\frac{1}{|\mathcal{V}|}\sum_{v\in\mathcal{V}}\mathbf{h}_v^{(L)} \;\Big\|\; \max_{v\in\mathcal{V}}\mathbf{h}_v^{(L)}\Big] \tag{8}$$

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log\,\hat{y}_{i,c} + \lambda\|\Theta\|_2^2 \tag{9}$$

Algorithm 1: GraphCSE forward pass

Input: program source x; embedding tables E_s, E_t;
layers {W_r, a_r,src, a_r,dst, W_o, g}_(l=1..L); head MLP

Output: class probabilities ŷ; graph embedding z_G

1: (V, {E_r}) ← BuildGraph(AST(x)) ▷ Eq. (1)

2: for each v ∈ V do

3: h_v^s ← E_s[τ(v)]; h_v^t ← E_t[σ(v)] ▷ Eq. (2)

4: α_v ← gate(h_v^s, h_v^t) ▷ Eq. (3)

5: h_v ← α_v W_s h_v^s + (1−α_v) W_t h_v^t ▷ Eq. (4)

6: end for

7: for l = 1 to L do

8: for each relation r and edge (u,v) ∈ E_r do

9: β_uv^r ← attention(h_u, h_v; r) ▷ Eq. (5)–(6)

10: end for

11: h_v ← LN(ELU(W_o h_v + Σ_r γ_r Σ_u β_uv^r W_r h_u)) ▷ Eq. (7)

12: end for

13: z_G ← [mean_v h_v ‖ max_v h_v] ▷ Eq. (8)

14: return softmax(MLP(z_G)), z_G

One layer costs O(|E| d + |V| d²) time for hidden width d and edge set E, and the whole model O(L(|E| d + |V| d²)), linear in program size; with d = 48 and L = 2 the network has roughly 132K parameters and trains on a single CPU core.

3.5 Open-set provenance scoring

A deployed detector must handle generators that were not in its training roster, so we attach rejection machinery to the trained model rather than to the label set. At the level of a single program we evaluate five standard rules on the frozen model: the closed-set rule (flag when the argmax lands on any non-human class); centroid distance and k-nearest-neighbor distance to the human training embeddings, both computed on z_G standardized by the human training moments, the latter a non-parametric density criterion in the tradition of Mahalanobis-style scoring [30]; the energy score, the negative log-sum-exp of the logits; and one minus the maximum softmax probability. Each score is thresholded at the 95th percentile of its human validation distribution, so every rule pays the same false-positive budget and no unseen data touches calibration.

Provenance decisions in practice rarely rest on one snippet: a submission history, a pull request, or a suspect repository supplies a set S of k programs from the same source. Sets carry information a single program cannot, because generator families are tight where humans are variable. We therefore test S with two statistics computed in the same standardized embedding space: the mean member k-NN distance to the human training bank, Eq. (10), and the within-set dispersion, Eq. (11). Both are calibrated by bootstrap over human validation programs at the same k, and S is flagged when either statistic is anomalous on a two-sided test, Eq. (12), with τ again the 95th percentile of human validation scores. The test is deliberately two-sided: a generator set can sit at anomalously high distance from the human bank (a style outside the envelope) or at anomalously low dispersion and distance (a style pinned to one point inside it), and both directions are evidence against human authorship.

$$m(S) = \frac{1}{k} \sum_{x \in S} \delta_5(\hat{\mathbf{z}}_x;\ \mathcal{H}_{\text{train}}) \tag{10}$$

$$d(S) = \frac{2}{k(k-1)} \sum_{x < x'} \|\hat{\mathbf{z}}_x - \hat{\mathbf{z}}_{x'}\|_2 \tag{11}$$

$$A(S) = \max\left(\frac{|m(S) - \mu_m|}{\sigma_m},\ \frac{|d(S) - \mu_d|}{\sigma_d}\right) > \tau \tag{12}$$

Nothing in Eqs. (10)-(12) is specific to GraphCSE; the same test runs in any vector space, and Section 5.4 uses exactly that to compare the transferability of GraphCSE, GCN, and raw TF-IDF representations under identical calibration.

4. Experimental Setup

4.1 Synthetic multi-family corpus

All experiments run on a corpus we generate ourselves; no program in it was written by a person or by a deployed language model, and every result in this paper should be read with that in mind. Each program is produced by a parameterized synthesizer that composes one to three small algorithmic tasks (sorting, searching, run-length encoding, and seventeen others) under a style profile with fifteen knobs: docstring probability and length, type-hint rate, identifier naming convention, comment density, comprehension-versus-loop preference, guard and exception density, helper factoring, f-string use, blank-line rhythm, and related surface habits. Every program is verified syntactically valid by the Python parser at generation time.

The class design encodes two facts about the real detection problem. Human code does not cover style space uniformly; programmers cluster into habitual styles. Our human class is therefore a mixture of eight fixed archetypes (a pragmatic scripter, a terse comprehension-heavy pythonista, a chatty tinkerer with dead code and TODO comments, and so on), each jittered per sample, which leaves gaps between the archetypes where a generator can plausibly sit. Second, individual generators phrase their documentation and comments in characteristic registers, the single strongest attribution cue in studies of real models [10]; each of our families draws docstrings and comments from its own pool, partially overlapping the human pool. Four families are used for training: GEN-V (verbose, documentation-heavy, type-hinted), GEN-T (template-uniform scaffolding), GEN-C (compact and idiomatic), and GEN-R (defensive, guard- and exception-heavy). GEN-C is deliberately parameterized to overlap the terse-pythonista human archetype, so the corpus contains, by construction, one family whose style barely leaves the human envelope; we use it to probe the boundary of what any detector can do. A fifth family, GEN-U, blends GEN-V documentation habits with GEN-C compactness under camelCase naming and is never used for training anywhere in the paper. Families sample tightly around their centroids (jitter 0.04-0.08 versus 0.10 for human archetypes), mimicking the low intra-model variance reported for real code generators [25].

Table 1. Corpus statistics per class: number of programs, mean lines of code, mean and standard deviation of graph nodes, and mean data-flow edges per graph.

Class	Programs	LOC (mean)	Nodes (mean ± std)	Data-flow edges (mean)
Human (8 archetypes)	500	15.5	90.3 ± 44.6	7.5
GEN-V	500	25.6	106.1 ± 47.8	7.8
GEN-T	500	16.8	102.5 ± 42.9	8.5
GEN-C (overlaps human)	500	10.7	77.9 ± 41	5.6
GEN-R	500	24.3	112.9 ± 49.4	7.5
GEN-U (unseen)	300	20	94.8 ± 44	6.3

Programs are short by design (Table 1), matching the snippet-level granularity at which detection tools are typically applied [5]. The five in-distribution classes are split 70/10/20 into train, validation, and test sets, stratified by class, with a fresh split per seed.

4.2 Baselines, protocols, and metrics

Four baselines cover the main representational alternatives. TF-IDF + LR applies logistic regression to character 3-5-gram TF-IDF vectors (20,000 features), the strongest classical stylometric approach in recent attribution studies [10]. StructMLP feeds a two-layer MLP with 150 hand-crafted features, the normalized AST node-type histogram plus surface statistics such as comment density and type-hint counts, standing in for classical feature-engineering pipelines [11]. BiGRU is a bidirectional GRU over the hashed lexical token sequence, the sequence-model counterpart of our graph encoder. GCN is a two-layer graph convolutional network [15] on the same graphs as GraphCSE but with untyped mean aggregation and fixed concatenation of the two channels, so the comparison isolates the heterogeneous components. All neural models use hidden width 48 and the same training budget; none received per-model hyperparameter tuning beyond the shared defaults, which keeps the comparison budget-fair, though tuning could plausibly move any of them by a point or two.

Three protocols structure the evaluation. In-distribution attribution (Section 5.1) is five-way classification over three seeds (11, 23, 47) governing splits, initialization, and batch order. Single-program open-set detection (Section 5.3) is leave-one-family-out: each of GEN-V/T/C/R is removed from training in turn, the model is retrained multi-class on the remaining classes over two seeds, and the held-out family plus the human test set are scored by the five rules of Section 3.5; GEN-U provides a fifth, fully external hold-out on top of the full five-class model. Set-level detection (Section 5.4) draws, for every k in {1, 3, 5, 10, 20}, 300 bootstrap sets of k programs from human validation data for calibration, then 300 sets each from the human test pool and from the held-out family, and applies the two-sided test of Eq. (12). Metrics are accuracy and macro-averaged F1 for attribution; detection rate at the fixed 5% validation false-positive budget, the realized test false-positive rate, and threshold-free AUROC for the open-set protocols. In total the paper rests on 67 neural training runs and 23 TF-IDF fits.

4.3 Implementation and reproducibility

Table 2. Hyperparameters and training configuration shared by all graph-based models.

Component	Setting
Hidden width d / layers L	48 / 2
Semantic hash buckets	512 (stable MD5 hashing)
Optimizer / learning rate / weight decay	Adam / 3×10^{-3} / 10^{-5}
Batch size / epochs	96 graphs / 18, best-validation checkpoint
Dropout (head)	0.2
Readout	concatenated mean + max
Open-set scoring	k = 5 nearest neighbors; 95th-pct validation threshold
Set-level test	B = 300 bootstrap sets; two-sided max-|z| statistic
Hardware / software	1 CPU core; Python 3.12, PyTorch 2.13, scikit-learn 1.7

Table 2 lists the configuration shared by the graph models and the open-set protocols. The full pipeline, corpus synthesis included, is deterministic given the seed and runs in roughly half an hour on one CPU core; a single GraphCSE training run takes roughly 45 seconds. Source code for the corpus generator, models, and every experiment accompanies the paper and is available from the corresponding author.

5. Results and Discussion

5.1 In-distribution multi-class attribution

Table 3. Five-way classification on the in-distribution test set, mean ± std over three seeds (%). Best value per column in bold.

Model	Accuracy	Macro-F1
GraphCSE (ours)	92.2 ± 0.6	92.1 ± 0.6
GCN	81.9 ± 2.8	80.9 ± 3.2
BiGRU	88.9 ± 0.9	88.6 ± 1.0
StructMLP	86.7 ± 1.4	86.3 ± 1.3
TF-IDF + LR	**93.6 ± 0.4**	**93.4 ± 0.5**

Table 3 reports the headline comparison, and we state its two messages in the order a skeptical reader would want them. First, the strongest in-distribution detector is not a neural model at all: character n-gram TF-IDF reaches 93.4 ± 0.5% macro-F1. With family-specific phrasing registers in the corpus, exactly as observed for real generators, where comment wording is the dominant cue [10], surface n-grams are a near-ideal instrument, and pretending otherwise would misrepresent the problem. Second, among learned representations the graph design matters a great deal: GraphCSE at 92.1 ± 0.6% sits 11.2 points above the GCN (80.9 ± 3.2%) on identical graphs and 3.6 above the BiGRU, with a third of the GCN's seed variance. The case for graph models over n-grams is not this table; it arrives in Section 5.4, where the representations, not the classifiers, are what transfers. Training behaves unremarkably (Figure 2), with GraphCSE converging faster and to a higher validation plateau than the GCN.

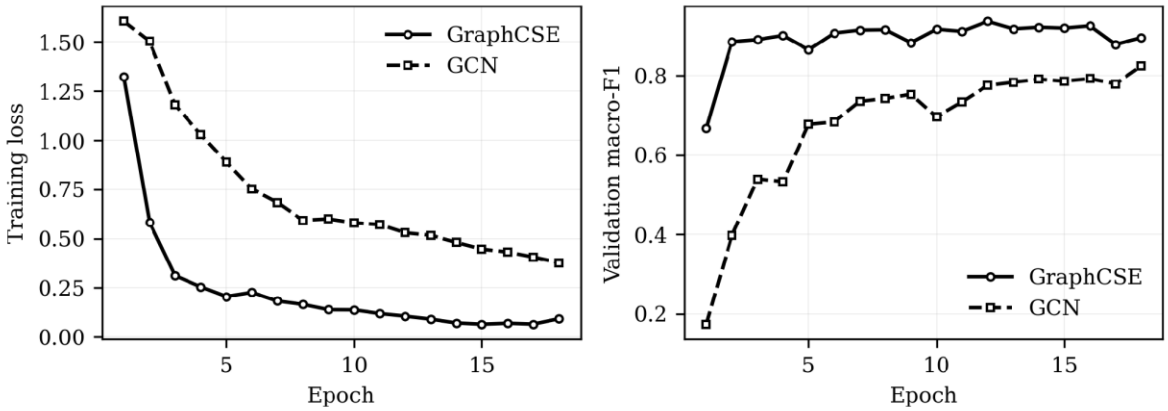

Figure 2. Training loss (left) and validation macro-F1 (right) for GraphCSE and the GCN baseline, seed 11.

The confusion structure (Figure 3) locates the residual error on the human class, whose recall is 79.2% against 92.1-100.0% for the generator families. Errors flow one way, from human archetypes into whichever family they abut, with the terse-pythonista archetype feeding the deliberately overlapping GEN-C. This asymmetry previews everything that follows: the human class is the wide one, and wide classes absorb.

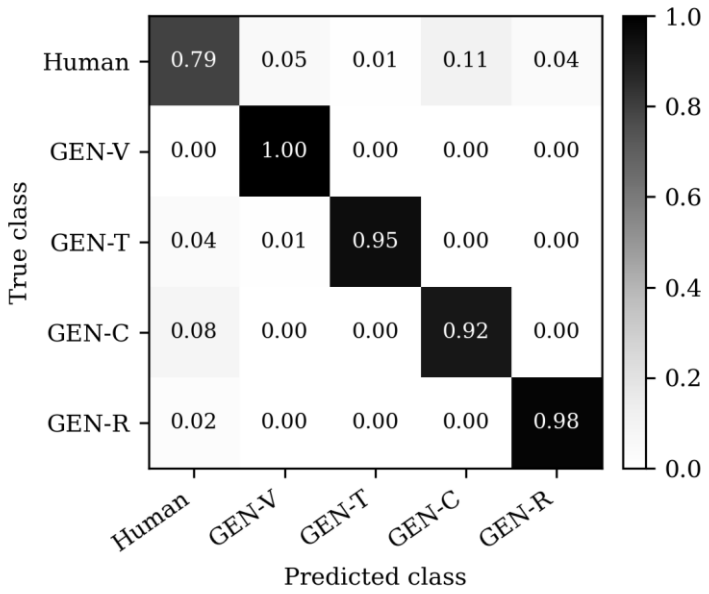

Figure 3. Row-normalized confusion matrix of GraphCSE on the test set, seed 11.

5.2 Ablations and the gate null result

Table 4. Ablations of GraphCSE components, macro-F1 mean ± std over three seeds (%), with the change relative to the full model. Best value in bold.

Variant	**Macro-F1**	**Δ vs. full**
Full GraphCSE	91.7 ± 0.8	—
Uniform fusion (α = 0.5)	**93.0 ± 0.3**	1.3
Structural channel only	82.4 ± 1.0	-9.3
Semantic channel only	92.1 ± 0.2	0.4
Single-relation attention	92.1 ± 0.4	0.3

The ablations (Table 4, Figure 4) contain a result that flatters the paper and one that does not, and both are reported at face value. Removing the semantic channel is catastrophic, 9.3 points, which is the corpus telling us again that phrasing dominates provenance. The unflattering finding concerns our own named mechanism: the learned HSSA gate does not outperform fixed uniform fusion, which is 1.3 points better on average, and the semantic channel alone essentially matches the full model. The gate values explain why (Figure 6): across every class the program-level mean gate settles in a narrow band around 0.49, so the network learns a global mixing ratio rather than the per-node, per-class specialization the design anticipated. We keep the gate in the architecture because it costs little and subsumes uniform fusion as a special case, but we do not claim it as a working mechanism on this corpus; whether node-level gating earns its parameters on real code, where the structural channel should matter more than our phrase-dominated corpus lets it, is an open question the synthetic setting cannot settle. Relation typing, for its part, contributes -0.3 points, small and within a standard deviation.

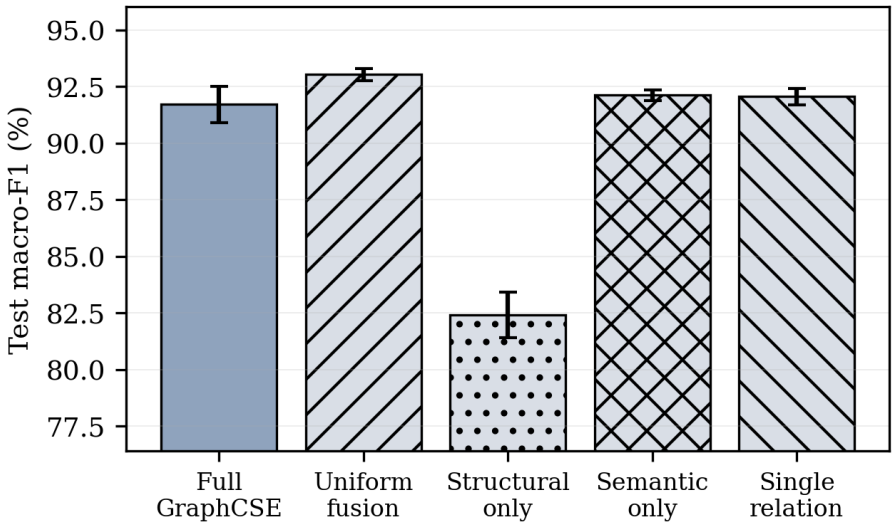

Figure 4. Ablation results, test macro-F1 with standard deviation over three seeds.

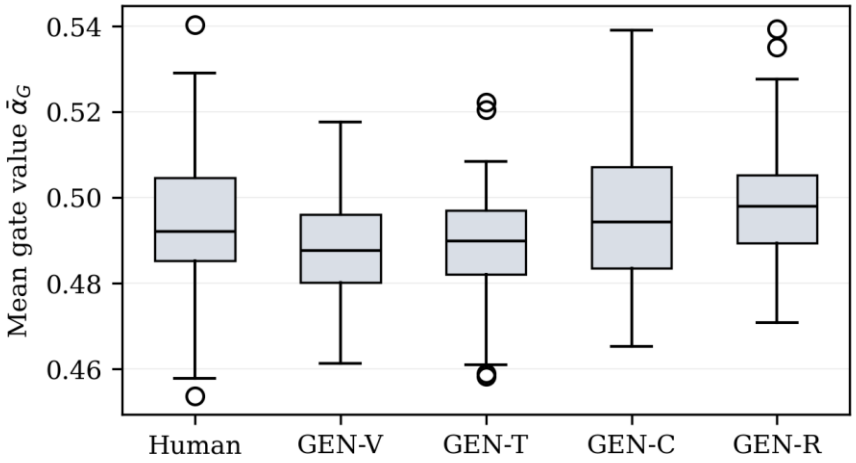

Figure 5. Distribution of program-level mean gate values $\bar{\alpha}$ by true class (test set, seed 11). The near-identical distributions across classes are the mechanism behind the uniform-fusion ablation result in Table 4.

5.3 Single programs from unseen generators: five ways to fail

Table 5. Closed-set (argmax) detection of held-out generator families: detection rate on the unseen family and false-positive rate on human test programs, mean over two seeds (%).

Held-out	GraphCSE Det.	GraphCSE FPR	GCN Det.	GCN FPR	TF-IDF Det.	TF-IDF FPR
GEN-V	16.5	22.3	15.5	34.7	13.8	27.2
GEN-T	1.9	19.3	56.4	36.1	0.1	17.8
GEN-C	3.2	6.9	7.5	18.8	1.3	8.9
GEN-R	11.6	18.8	15.2	31.2	2.8	26.2
GEN-U	66.3	20.8	51.0	55.9	44.7	26.7

Mean	19.9	17.6	29.1	35.3	12.5	21.4

Closed-set prediction is unreliable in a way no averaging can hide (Table 5). Mean detection sits at 19.9% for GraphCSE, 29.1% for the GCN, and 12.5% for TF-IDF, at false-positive rates of 17-33%, and the per-family spread is chaotic: the same GraphCSE that flags 66.3% of GEN-U programs flags 1.9% of GEN-T. An unfamiliar style is, to a closed-set classifier, simply another strange human, and which side of a decision boundary it lands on is an accident of geometry rather than a measurement of anything.

Table 6. Single-program score-based rejection: AUROC of each scoring rule on held-out family versus human test programs, mean over two seeds. Values near 0.5 are chance; values below 0.5 mean the unseen family looks more human-typical than humans.

Held-out	**GCSE k-NN**	**GCSE energy**	**GCSE MSP**	**GCN k-NN**	**GCN energy**	**TF-IDF dist.**
GEN-V	0.48	0.83	0.62	0.40	0.74	0.88
GEN-T	0.37	0.42	0.32	0.36	0.56	0.48
GEN-C	0.60	0.50	0.45	0.67	0.30	0.66
GEN-R	0.72	0.67	0.55	0.62	0.55	0.66
GEN-U	0.81	0.83	0.76	0.72	0.44	0.76
Mean	0.59	0.65	0.54	0.55	0.52	0.69

Score-based rejection does no better (Table 6). Across centroid distance, k-NN distance, energy, and maximum softmax probability, mean AUROC never exceeds 0.65 for any model, detection at the 5% budget stays in single digits to low tens, and several cells sit below 0.5, meaning the unseen family is more human-typical, by that score, than humans themselves. The pattern survives the change of scoring paradigm because its cause is upstream of scoring: a generator family is a tight cluster, and when that cluster lies inside the envelope spanned by human archetypes, no function of a single embedding can separate membership in it from ordinary human variation. We regard this as the central negative result of the paper, and we note that it is invisible to any evaluation that only tests generators from the training roster, which is the standard protocol in the detection literature [5], [10]. The one partial exception proves the rule: GEN-V, whose verbose register pushes it furthest from the human pool, is the only family any single-program score handles respectably (TF-IDF distance 0.88, GraphCSE energy 0.83 AUROC).

5.4 Sets of programs: model-agnostic detection recovered

The set-level test of Section 3.5 changes the outcome qualitatively. Figure 5 sweeps the set size k, and Table 7 details k = 10; at k = 1 the test reduces to single-program k-NN rejection and inherits its failure, which is the honest left edge of the curve.

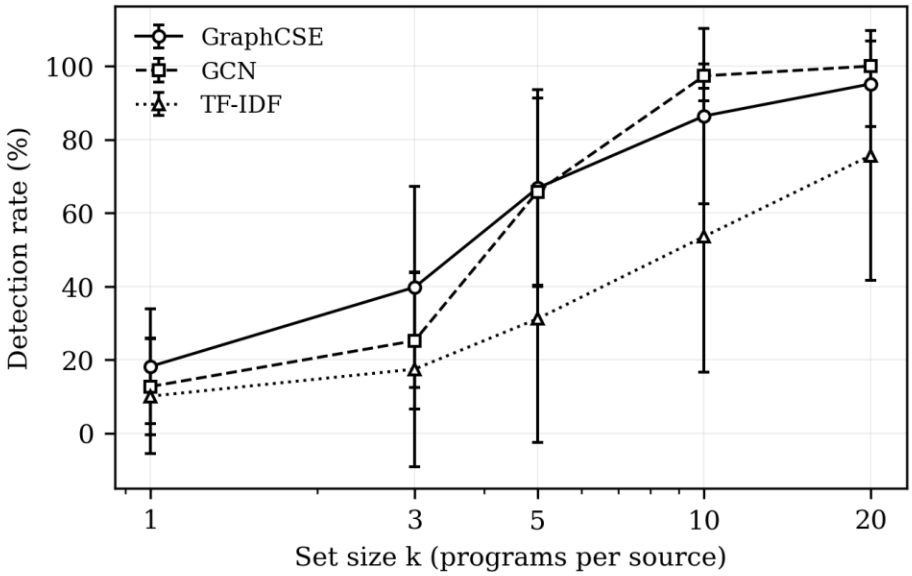

Figure 6. Set-level detection rate at the 5% validation false-positive budget versus set size k, mean and std over five held-out families and two seeds. k = 1 coincides with single-program rejection.

Table 7. Set-level detection at k = 10 programs per source: detection rate at the 5% validation false-positive budget, realized test false-positive rate, and AUROC, mean over two seeds. Best detection per row in bold.

Held-out	GCSE Det.	FPR	AUROC	GCN Det.	FPR	AUROC	TF-IDF Det.	FPR	AUROC
GEN-V	95.8	12.3	0.98	96.3	11.5	0.98	**99.7**	13.7	1.00
GEN-T	98.5	16.7	0.99	**100.0**	16.0	1.00	11.5	12.2	0.58
GEN-C	40.7	8.5	0.81	**97.2**	23.0	0.95	56.8	5.5	0.82
GEN-R	97.0	27.2	0.92	**97.8**	20.0	0.94	29.0	10.7	0.66
GEN-U	**100.0**	15.0	1.00	95.3	12.2	0.98	71.0	10.5	0.91
Mean	86.4	15.9	0.94	**97.3**	16.5	0.97	53.6	10.5	0.79

Three observations. First, the recovery is large and monotone: mean detection in the GraphCSE embedding climbs from 18.2% at k = 1 to 86.4% at k = 10 and 95.1% at k = 20, and even GEN-T, undetectable to every single-program rule, reaches 98.5% because a template family is pathologically tight and the dispersion statistic of Eq. (11) sees it. Second, the learned graph embeddings are what make the test work: the identical procedure in raw TF-IDF space manages 53.6% at k = 10 (AUROC 0.794), so the representation that loses the in-distribution race in Table 3 wins the one that matters for unseen generators. Third, a finding we did not anticipate and report against our own model's interest: the GCN embedding is the better rejection space, 97.3% mean detection (AUROC 0.972) versus GraphCSE's 86.4%, with the gap concentrated on the human-overlapping GEN-C (97.2% versus 40.7%). Supervised accuracy and open-set transferability are evidently different axes: GraphCSE's sharper class separation appears to fold near-human styles into the human region, while the weaker GCN preserves more of the stylistic variance that the anomaly test needs. Detection is costly at the boundary in either space, and GEN-C at 40.7-97.2% remains the honest floor of the method.

One calibration caveat belongs next to these numbers rather than in a footnote. The realized false-positive rates in Table 7 run above the 5% design budget, 15.9-16.5% at k = 10 and higher at k = 20, because bootstrap sets drawn from roughly one hundred human validation programs overlap heavily at large k, which understates the variance of

the set statistics and tightens τ. The AUROC columns are free of this artifact, and a deployment with more human calibration data than our deliberately small validation pool would not face it at these set sizes; we report the inflated rates as measured.

5.5 Limitations

Four limitations bound the claims. First and most important, the corpus is synthetic. Our families are stylistic constructions with controlled variance, not samples from deployed models; real LLM output entangles task-conditioned semantics with style in ways our synthesizers do not reproduce, and transfer of these results to production detectors is an empirical question this paper does not answer. The corpus design is itself a finding of sorts, and we disclose it: an earlier version of this study drew human style knobs independently and uniformly, which makes the human envelope cover all of style space and open-set detection provably hopeless; results from that design are not reported, and the archetype mixture used here is our attempt at a fairer, gap-containing model of human variation. Second, programs are short, single-file Python; repository-scale artifacts, other languages, and adversarially post-edited code are outside the evaluation, and stylometric detectors are known to degrade under exactly such transformations. Third, the set-level test assumes the k programs share a source; mixed sets, a human project with partial AI assistance being the practically urgent case, would dilute both statistics, and characterizing that dilution is future work rather than a solved problem. Fourth, the calibration inflation quantified in Section 5.4 means the fixed-budget detection rates at large k should be read alongside their realized false-positive rates, not in isolation.

6. Conclusion

We set out to build a multi-class detector of AI-generated programs and to test the model-agnostic claim that usually accompanies such systems. The detector part worked: GraphCSE, a four-relation code graph encoder with dual structural-semantic channels, is the strongest learned model on our benchmark at 92.1 ± 0.6% macro-F1, though plain character n-grams edge it in-distribution and our own gating mechanism proved no better than uniform fusion, two results we report as measured. The model-agnostic part reshaped the paper. Single programs from unseen generator families defeat every standard rejection rule we tested, not marginally but structurally, because a tight unseen style inside the human envelope is indistinguishable from an unusual human; and the capability returns only when detection is posed at the level of program sets, where a two-sided anomaly test on embedding statistics reaches 86.4-97.3% mean detection of five unseen families at ten programs per source, with the learned graph embeddings, not raw n-gram space, supplying the transferable geometry. The practical reading is direct: claims of model-agnostic code detection should be evaluated open-set and stated at the collection level, and the natural next steps, validating the set-level test on real generator corpora, extending it to mixed human-AI sets, and training embeddings whose human region is deliberately compact, all start from the boundary this study maps.

REFERENCES

1. M. Chen et al., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.
2. B. Rozière et al., "Code Llama: Open foundation models for code," arXiv preprint arXiv:2308.12950, 2023.
3. R. Li et al., "StarCoder: May the source be with you!," Transactions on Machine Learning Research, 2023.
4. W. H. Pan et al., "Assessing AI detectors in identifying AI-generated code: Implications for education," in Proc. IEEE/ACM 46th Int. Conf. Software Engineering: Software Engineering Education and Training (ICSE-SEET), 2024, pp. 1–11.
5. P. T. Nguyen, J. Di Rocco, C. Di Sipio, R. Rubei, D. Di Ruscio, and M. Di Penta, "GPTSniffer: A CodeBERT-based classifier to detect source code written by ChatGPT," Journal of Systems and Software, vol. 214, p. 112059, 2024.
6. X. Yang, K. Zhang, H. Chen, L. Petzold, W. Y. Wang, and W. Cheng, "Zero-shot detection of machine-generated codes," arXiv preprint arXiv:2310.05103, 2023.
7. E. Mitchell, Y. Lee, A. Khazatsky, C. D. Manning, and C. Finn, "DetectGPT: Zero-shot machine-generated text detection using probability curvature," in Proc. 40th Int. Conf. Machine Learning (ICML), 2023, pp. 24950–24962.
8. J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, "A watermark for large language models," in Proc. 40th Int. Conf. Machine Learning (ICML), 2023, pp. 17061–17084.
9. V. S. Sadasivan, A. Kumar, S. Balasubramanian, W. Wang, and S. Feizi, "Can AI-generated text be reliably detected?," arXiv preprint arXiv:2303.11156, 2023.

10. T. Bisztray et al., "I know which LLM wrote your code last summer: LLM generated code stylometry for authorship attribution," in Proc. 18th ACM Workshop on Artificial Intelligence and Security (AISec), 2025.
11. A. Caliskan-Islam et al., "De-anonymizing programmers via code stylometry," in Proc. 24th USENIX Security Symposium, 2015, pp. 255–270.
12. M. Abuhamad, T. AbuHmed, A. Mohaisen, and D. Nyang, "Large-scale and language-oblivious code authorship identification," in Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS), 2018, pp. 101–114.
13. M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in Proc. Int. Conf. Learning Representations (ICLR), 2018.
14. U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," Proceedings of the ACM on Programming Languages, vol. 3, no. POPL, pp. 1–29, 2019.
15. T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in Proc. Int. Conf. Learning Representations (ICLR), 2017.
16. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in Proc. Int. Conf. Learning Representations (ICLR), 2018.
17. X. Wang et al., "Heterogeneous graph attention network," in Proc. World Wide Web Conf. (WWW), 2019, pp. 2022–2032.
18. M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in Proc. European Semantic Web Conf. (ESWC), 2018, pp. 593–607.
19. Z. Feng et al., "CodeBERT: A pre-trained model for programming and natural languages," in Findings of the Association for Computational Linguistics: EMNLP, 2020, pp. 1536–1547.
20. D. Guo et al., "GraphCodeBERT: Pre-training code representations with data flow," in Proc. Int. Conf. Learning Representations (ICLR), 2021.
21. Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), 2021, pp. 8696–8708.
22. Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 32, 2019.
23. Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in Proc. Int. Conf. Learning Representations (ICLR), 2016.
24. Z. Xu and V. S. Sheng, "Detecting AI-generated code assignments using perplexity of large language models," in Proc. AAAI Conf. Artificial Intelligence, vol. 38, no. 21, 2024, pp. 23155–23162.
25. T. Ye et al., "Uncovering LLM-generated code: A zero-shot synthetic code detector via code rewriting," arXiv preprint arXiv:2405.16133, 2024.
26. J. Xu et al., "Investigating efficacy of perplexity in detecting LLM-generated code," arXiv preprint arXiv:2412.16525, 2024.
27. I. Solaiman et al., "Release strategies and the social impacts of language models," arXiv preprint arXiv:1908.09203, 2019.
28. S. Gehrmann, H. Strobelt, and A. M. Rush, "GLTR: Statistical detection and visualization of generated text," in Proc. 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2019, pp. 111–116.
29. A. Uchendu, T. Le, K. Shu, and D. Lee, "Authorship attribution for neural text generation," in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 8384–8395.
30. K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 31, 2018.