FROM TOKENS TO FILES: MULTI-GRANULARITY DETECTION AND LOCALIZATION OF AI-INVOLVED CODE

Anonymous authors
Paper under double-blind review

ABSTRACT

The increasing use of large language models for code generation has created an urgent need for reliable methods to distinguish purely human-written code from *AI-involved* code — files in which any part was generated or substantially modified by a model, however small that part is. Our goal is not only to determine whether a program is purely human-written or has been generated or modified by AI, but also to localize the specific lines or blocks attributable to AI. Existing approaches typically compress a file into a single score, either from surface-level stylistic features or from a global code representation, which averages away the evidence when only a few blocks of a file are AI-involved. We propose a detection framework that models AI authorship at the token level and supports predictions at the line, block, and file levels. A frozen code language model is equipped with two low-rank adapters, trained separately on machine-written and human-written files. The per-token log-likelihood ratio between the two adapters provides local evidence of AI authorship. Lightweight prediction heads then aggregate this evidence within lines and blocks, using the surrounding code context to estimate the probability that each line or block was generated or substantially modified by AI. The resulting local predictions are further aggregated to produce a file-level decision. We train these predictions using mixed-authorship files constructed from human originals and their AI-generated or AI-modified counterparts, including continuation and fill-in-the-middle scenarios. On an internal test set of $11{,}773$ AI-involved files across six generation regimes against a shared pool of $2{,}541$ human files, our method reaches 72.4 macro F1 at a 1% FPR operating point. With no adaptation at all, the method reaches 62.5 line-level F1 on HybridCodeAuthorship (Patterson et al., 2026), an external human–AI co-writing benchmark that ships no training split, above the 56 that the benchmark's own paper reports as its best line-level result. We further run an extra transfer evaluation on a human-written-only test set: across $19{,}445$ files from repositories created before 2022, our false-positive rate stays between 0.55 and 0.81% when the false-positive rate threshold is transported at 1%. So the operating point remains valid after transferring from paired synthetic negatives to real human code.

1 INTRODUCTION

Assistants that write code inside an existing file have moved the unit of authorship below the file. A commit may contain a completed function body, a refilled interior span, or a locally refactored block, with the rest of the file untouched. The question a provenance tool has to answer is therefore not only *was this file written by a model* but *is any part of it*, and, if a reviewer is to act on the answer, *which part*.

Two families of detectors dominate the current literature, and both answer only the first question. Zero-shot likelihood statistics score a file against a reference language model and threshold a scalar summary: log-rank (Gehrmann et al., 2019), LRR (Su et al., 2023), perturbation curvature (Mitchell et al., 2023; Bao et al., 2024), or cross-perplexity between two models (Hans et al., 2024). Supervised detectors fine-tune a code encoder (Feng et al., 2020; Guo et al., 2022; Wang et al., 2023*a*) or align a proxy model to machine style (Chen et al., 2025) and read a single decision out of a global representation. In both cases the file is compressed to one number before any decision is made, which is adequate when the whole file is machine written and lossy when only a few blocks are.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Mixed authorship breaks these detectors in two distinct ways. A global score averages a short AI span together with the human majority of the file, so the evidence is diluted in proportion to the AI fraction. Independently, the span itself is weaker evidence than a whole machine-written file, because the model that produced it was conditioned on the surrounding human code and inherited its naming and layout. Our measurements show both effects: on the two interior-segment test cells, the strongest supervised file-level baseline drops to 0.647 and 0.565 AUROC, while the same detector reaches 0.945–0.963 on the from-scratch cells.

We therefore keep the evidence local for as long as possible. A single frozen code language model is equipped with two LoRA adapters (Hu et al., 2022), trained with a plain causal language modeling objective on machine-written and on human-written files respectively; neither has a classification head. Their per-token log-likelihood ratio is a local score whose human-side term cancels the confound of how predictable a token is in the first place.

Evaluating this design requires separating what a detector cannot represent from what its training distribution never contained. The transfer–adaptation gap on external benchmarks is the sharpest finding in the paper, and we read it as a statement about training corpora rather than detector capacity.

The main contributions of this paper are:

- A detection framework that couples a token-level likelihood ratio with line- and block-level supervised heads over a single frozen code language model, so that local generation evidence and its surrounding context are modeled jointly rather than as alternatives.
- A mixed-authorship data construction protocol that synthesizes continuation and fill-in-the-middle edits with character-level ground truth and a controlled AI ratio ρ, which supplies within-file supervision for the line and block heads and test cells covering both tail and interior segment geometries. Appendix A.5 ablates the training-side composition and finds corpus volume, rather than the mixture of forms, to be the load-bearing factor.
- An evaluation protocol that separates *capacity* from *distribution* on external benchmarks under two protocols (zero-adaptation transfer and per-benchmark adaptation), reporting results at the file, block and line granularities, together with a carrier-artifact control and a false-positive measurement on real repository code.

We defer a full discussion of related work to Appendix A.1.

2 Problem Formulation

Let $x = (x_1, \ldots, x_T)$ be the token sequence of a source file. The file-level label is binary and *AI-involved*: $y = 1$ if any part of the file was generated by a language model, and $y = 0$ if the file is entirely human-written. Mixed files therefore carry $y = 1$ regardless of how little of the file is machine-written, which is the convention used in machine-revised-text detection (Chen et al., 2025) and by the *machine-refined* class of DroidCollection (Orel et al., 2025). A detector scores the file and thresholds that score at τ, calibrated on human files at a target false-positive rate. The detector sees the file alone at test time: no repository metadata, commit history or generation-side information is available.

Ground truth for a mixed file is a set of character intervals covering its machine-written text. A line is an *AI line* if any of its characters falls inside one of those intervals. The AI ratio ρ is the fraction of AI lines in the file, over all lines including blank ones; our synthesized mixed files carry $\rho \geq 0.05$ by construction (Section 3.4) and pure AI files have $\rho = 1$. A *block* is a run of K consecutive lines with $K \in \{8, 16\}$, cut from the first line of the file; the last block keeps whatever fewer than K lines remain, since dropping it would exclude the end of the file from prediction and the machine-written tail of continuation files sits exactly there. Block b carries the label $z_b = 1$ if it contains at least one AI line and $z_b = 0$ otherwise, so a block whose AI lines cover only part of it is positive while being mixed in content. Blocks and lines are the two prediction units we report. Every line-level number in the paper is computed against these line labels, and every block-level number against these block labels except where a caption states the convention it departs to.

3 METHOD

3.1 TOKEN-LEVEL LIKELIHOOD RATIO

We keep a single code language model p_θ frozen and train two low-rank adapters (Hu et al., 2022) on it with a plain causal language modeling objective: an AI-side adapter on machine-written files and a human-side adapter on human-written files. Neither adapter has a classification head. The per-token score is the log-likelihood ratio

$$s_t \;=\; \log p_{\mathrm{ai}}(x_t \mid x_{<t}) \;-\; \log p_{\mathrm{hum}}(x_t \mid x_{<t}), \tag{1}$$

which we decompose through the frozen base model as $s_{\text{ai-hum}} = s_{\text{ai-base}} + s_{\text{base-hum}}$, where $s_{\text{ai-base}} = \log p_{\mathrm{ai}} - \log p_\theta$ and $s_{\text{base-hum}} = \log p_\theta - \log p_{\mathrm{hum}}$ are the two adapters' deviations from the shared base model. Neither half is sufficient on its own: subtracting the human-side term cancels the confound of *how predictable the text is in the first place*, which a single-sided score cannot separate from authorship (Section 5.3).

Both adapters are rank-16 LoRA modules with $\alpha = 32$ (18.5M trainable parameters per side, 36.9M in total), trained for one epoch in fp16 on the pure-file split, with no packing and token-budget batching. The frozen backbone is Qwen2.5-Coder-1.5B (Hui et al., 2024).

3.2 LINE- AND BLOCK-LEVEL DISCRIMINATIVE HEADS

Token scores are mapped to lines (a token belongs to the line containing its last character) and then to fixed-K blocks. Each block is described by 88 features in five families: raw statistics of s_t, of the base-model profile (negative log-likelihood, entropy, rank) and of dctx, the gap between the long- and short-context negative log-likelihood of the same token under each of the three passes; within-file z-scores of those statistics; neighbouring-block contrasts; positional features; and language identity (Table 21). A sequence head (bidirectional GRU; Cho et al., 2014) reads the block sequence of a file and emits a per-block AI probability $\hat{z}_b$. The head is trained only on blocks that are entirely AI or entirely human, while reading the whole file, mixed blocks included, as context; at inference it scores every block, and block-level metrics are computed against z_b over all blocks.

The line head is the same construction one granularity finer: a bidirectional GRU over the per-token signal sequence, mean-pooled to one logit and one AI probability $\hat{z}_\ell$ per line. Line labels come directly from the AI intervals, so no line has to be held out of the loss. Both heads consume the same token-level profile, each signal paired with its within-file z-score, and differ in where it is pooled, the block head into per-block statistics before the sequence model and the line head only after it, as well as in their feature sets (Appendix A.8). Which granularity is preferable depends on the regime, so we report both: Section 5.2 gives their behaviour against AI segment length; a syntax-aligned alternative to the fixed grid and the two heads at equal input are in Appendix A.5.

3.3 AGGREGATION TO A FILE-LEVEL SCORE

Each read-out produces a file-level score $f(x)$ by averaging its own units: the token ratio over the T tokens of the file, the block head over the logits of the block set $\mathcal{B}$, and the line head over the logits of the L lines,

$$f_{\mathrm{lr}}(x) = \frac{1}{T}\sum_{t=1}^{T} s_t, \quad f_{\mathrm{blk}}(x) = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}} \sigma^{-1}(\hat{z}_b), \quad f_{\mathrm{lin}}(x) = \frac{1}{L}\sum_{\ell=1}^{L} \sigma^{-1}(\hat{z}_\ell), \tag{2}$$

where σ^{-1} is the logit function. The mean is the selected form in all three cases and not the only one we tried: for the token score we compare it against a sliding-window maximum, a change-point tail mean and the best segment after binary segmentation (Truong et al., 2020; Killick et al., 2012), and for each head against a per-unit maximum and a learned attention aggregator over unit states. The aggregator and the block size K are selected on the development split and applied to test unchanged.

Thresholds are calibrated as stated in Section 4.4, except where a table states otherwise: the block and line read-outs use thresholds frozen on the development split and transported unchanged, both at their own granularity and, in Section 5.5, to the pre-2022 repository corpus, which is disjoint from the internal paired HUMAN cell.

3.4 CONSTRUCTING MIXED-AUTHORSHIP SUPERVISION

Supervision at either granularity requires files whose authorship is heterogeneous *and* known. We synthesize two forms. In **continuation** a model completes a file from its first $N\%$ of lines, which gives exactly one AI segment at the tail. In **fill-in-the-middle** one or more spans are removed and refilled, which gives interior segments (Bavarian et al., 2022); the number of holes is sampled from $\{1, 2, 3, 5, 8\}$. Sampling is constrained so that ρ is balanced over five buckets covering $[0.05, 1.0]$, with the cut point or the hole coverage solved backwards from the target ρ and resampled when the realized ratio falls outside its bucket. AI spans are recorded as exact whole-line character intervals. For every mixed file we additionally keep its original human version as a negative example, so the heads see "no boundary anywhere" files that are otherwise textually paired with positives.

Two properties of this construction bound what the heads can learn. Both forms produce long segments — the median AI segment in the training and development splits is 20 non-blank lines — so the scattered short-segment regime that in-place refactoring produces is absent from training and appears only on the test side, where we read it as out-of-distribution measurement rather than as a held-out sample of the training regime. The dual-LoRA backbone, by contrast, is trained on pure files only. The development split used for aggregation and threshold selection does contain mixed files, because the score of a file moves with its AI ratio ρ: pure files sit at the extremes, one mode per class, while a partially machine-written file falls between them. A development split of pure files alone would fix the aggregator choice at scores the mixed part of the test set almost never produces.

4 EXPERIMENTAL SETUP

4.1 INTERNAL DATASET

Files are drawn from public repositories and paired with machine-written counterparts produced by a two-stage protocol in which a reader model summarizes a file into a specification and a writer model reimplements the file from the specification alone, without seeing the original. The isolation is deliberate: a writer that sees the original produces *a rewrite*, and a detector trained on rewrites learns rewriting artifacts. The reader is GPT-5.6 decoded at temperature 0; the writers are Claude Opus 5, DeepSeek-V4-Flash, GPT-5.6 and GLM-5.2, all decoded at temperature 0.7, and every file is assigned to one of them at random so that each contributes about a quarter of the machine-written side. Mixing writers dilutes single-model length and comment-density fingerprints. Splits are cut by repository hash, so no repository appears on both sides of a split.

Table 20 (Appendix A.8) summarizes the splits: the two LoRA adapters train on 12,002 pure human/AI files, the block head on 1,582 mixed files (843 continuation and 739 fill-in-the-middle) together with their 1,582 human originals, and the aggregator, K and decision thresholds are fixed on 4,062 pure and 398 mixed development files. The test set applies one generation axis at a time to a fixed human base — 11,773 positives over six generation cells against a shared pool of 2,541 human negatives.

4.2 TEST CELLS

The test set varies one axis at a time over a fixed human base: HUMAN (the only negative cell), BASELINE (specification-based rewriting with the training writer models), UNSEEN-MODELS (same protocol with four writer models never seen in training: Qwen3-Coder-Plus, Doubao-Seed-2.1-Pro, Kimi-K2.6 and MiniMax-M2.7), FULL-REWRITE (the model sees the original and writes its own implementation), COMPLETION (a human prefix is completed), REFACTOR (in-place refactoring, producing many very short AI segments), and FIM (interior spans refilled). The cells differ in how much human text survives in the output, which we find to be the dominant difficulty factor.

4.3 BASELINES

Every supervised baseline shares our training split, the same development split for aggregation and threshold selection, and, where applicable, the same scoring backbone. **Group A (zero-shot)** comprises six likelihood-based statistics: log-rank (Gehrmann et al., 2019), LRR (Su et al., 2023), Fast-DetectGPT (Bao et al., 2024), Binoculars (Hans et al., 2024), DetectCodeGPT (Shi et al., 2025)

and the AIGCode Detector (Xu and Sheng, 2024). **Group B (supervised)** comprises a frozen code encoder (CodeT5+ 220m) with an MLP head, fully fine-tuned CodeBERT (Feng et al., 2020) and UniXcoder (Guo et al., 2022), ImBD (Chen et al., 2025), DroidDetect (Orel et al., 2025), and — the critical control — *the same backbone with a LoRA classification head*, matched in backbone, data and optimisation schedule, so that the only difference from our method is "classification head versus likelihood ratio". **Group C (ours)** is the token-level likelihood ratio, the block head at $K \in \{8, 16\}$, the line head, and their combinations. Scoring and aggregation details are in Appendix A.9.

4.4 METRICS

The positive class is *AI-involved*, so a mixed file is positive whatever its ρ (Section 2). We report AUROC, TPR@1%FPR and macro F1 over the positive and negative files of each comparison, scored on the file score f. An AUROC below 0.500 means the score ranks machine files as *more* human-like than human files. **TPR@1%FPR** reads its threshold off the negatives of the same comparison; on the internal test set that is the HUMAN cell of the test set itself, so the operating point is exact rather than transported from development. **Macro F1** averages the per-class F1 of the AI class and of the human class, the latter scored on the inverted prediction, which penalizes a detector that calls everything AI-involved; it needs a threshold and the available calibration data differ by corpus, so each table states its own. Accuracy appears only in Appendix A.2: the fraction of mixed files differs between development and test, so a transported threshold is systematically mismatched, which is why the three metrics above are either threshold-free or read at a fixed FPR. Every table in the paper follows the same conventions for metrics, negative pool, chance levels and bolding, and states only its departures from them; table conventions, chance levels, per-cell counts and per-row exceptions are in Appendix A.9.

4.5 EXTERNAL BENCHMARKS AND TRANSFER PROTOCOLS

We evaluate on three external resources, none of which shares our generation protocol: **AICD-Bench** (Orel et al., 2026), used as its binary task T1 and its four-class task T3; **DroidCollection** (Orel et al., 2025), a multi-generator, multi-task corpus with four authorship classes; and **Hybrid-CodeAuthorship** (HCA) (Patterson et al., 2026), the only one of the three with per-line ground truth. For the multi-class resources every machine-involved class is mapped to $y = 1$. Two protocols are reported: **P1** transfer with no adaptation; and **P2** adaptation on the benchmark's own training split, which is the protocol under which published numbers were obtained. HCA has no training split, so only P1 is defined there, and it is additionally scored at its native line and block granularity. Group A and the Group B baselines are run on the external benchmarks as well. Composition, file counts, per-slice results and the row set of each table are in Appendix A.10.

5 RESULTS

5.1 MAIN COMPARISON

Group A sits at the all-negative floor of the operating point: on the three from-scratch cells (BASE, UNSEEN, REWRITE) the six likelihood statistics are within 2.1 points of it, because TPR at 1% FPR is essentially zero there. This is not a scoring-direction error — flipping Binoculars or Fast-DetectGPT does not recover a usable detector — it is the assumption "machine code is more predictable than human code" failing on specification-based rewriting, whose writer never sees the original. Where human text is left in place, the zero-shot statistics leave the floor only marginally: 38.9–41.8 on COMPL. (floor 38.6), 40.1–40.9 on FIM (39.8) and 36.4–36.7 on REFAC. (36.4).

Group B changes the picture completely: every supervised baseline reaches 53.1–82.9 on the from-scratch cells, against at most 35.9 for any zero-shot statistic, so what makes specification-based rewriting detectable is supervision on this distribution, not a better likelihood statistic. The informative comparison is the same-backbone LoRA classification head, which shares the backbone, the training files, the tokenizer and the optimisation schedule with our adapters and, at scoring time, the same 4096-token window with 512 tokens of carried context as our own scorer (its file score is the mean over window logits), so its residual gap to the token ratio (60.4 against 70.7 average macro F1) is attributable to the likelihood-ratio formulation rather than to the 1.5B backbone. The gap has

Table 1: File-level detection on the internal test set. Per-cell entries are macro F1 (%) at the 1% FPR threshold read off the HUMAN cell; at that threshold a score without signal labels almost every file human, and the all-negative predictor scores 33.3–39.8%, which is the floor each column has to clear. *Avg.* is the unweighted mean over the six cells, given for all three metrics. Group C rows are cumulative, not independent detectors. Every cell of this table with AUROC and TPR@1%FPR as well is Table 22 (Appendix A.9).

	Method	BASE	UNSEEN	REWRITE	COMPL.	REFAC.	FIM	Avg.
		macro F1 (%) at the 1% FPR operating point						AUROC / TPR@1%FPR / F1
A	log-rank	33.3	33.2	35.7	39.1	36.5	40.9	0.488 / 0.4 / 36.5
	LRR	33.4	33.2	35.7	38.9	36.4	40.3	0.481 / 0.3 / 36.3
	Fast-DetectGPT	33.3	33.3	35.9	41.8	36.6	40.1	0.481 / 0.8 / 36.8
	Binoculars	33.2	33.2	35.7	39.1	36.7	40.4	0.488 / 0.4 / 36.4
	DetectCodeGPT	33.3	33.1	35.7	39.1	36.5	40.6	0.518 / 0.4 / 36.4
	AIGCode Detector	34.1	33.8	33.8	34.3	33.2	34.7	0.475 / 0.9 / 34.0
B	CodeT5+ + MLP	63.6	61.7	58.2	40.5	43.4	41.1	0.771 / 17.3 / 51.4
	CodeBERT	72.2	64.9	69.1	40.6	50.6	41.6	0.791 / 24.4 / 56.5
	UniXcoder	74.7	68.5	72.5	41.8	51.5	41.7	0.817 / 27.3 / 58.4
	ImBD	57.0	57.8	53.1	40.8	40.8	41.3	0.723 / 13.4 / 48.5
	Same backbone + LoRA cls. head	79.3	71.6	75.0	42.2	52.9	41.3	0.808 / 30.5 / 60.4
	DroidDetect	82.9	80.9	78.6	43.7	54.9	42.2	0.801 / 36.1 / 63.8
C	Ours: token ratio, file mean	**89.1**	89.4	**87.4**	50.0	63.1	45.3	0.838 / 47.1 / 70.7
	+ block head, $K = 8$	86.3	90.7	83.3	**60.8**	62.9	**50.4**	0.852 / **48.4** / **72.4**
	+ block head, $K = 16$	87.1	**91.6**	83.7	58.5	**63.8**	48.6	**0.853** / **48.4** / 72.2
	+ line head, file mean	79.8	82.9	72.4	57.1	54.8	47.1	0.836 / 37.4 / 65.7

two parts. On the from-scratch cells (9.8–17.8 points) it is a tail effect: at the same 1% FPR the token ratio recovers more positives. On COMPL. and FIM it is structural: the classifier stays 3.6 and 1.5 points above the all-negative floor, against 11.4 and 5.5 for the token ratio. A windowed head compresses each window into one decision and averages the windows, so a short inserted span is averaged away, whereas a token-level score keeps it addressable.

Difficulty tracks how much human text survives in the output: REWRITE (87.4) is as easy as the specification-based cells (89.1, 89.4), while COMPL. and FIM, where a human prefix or envelope is left in place, are the two hardest (50.0 and 45.3). The block head's gain is concentrated exactly there (+10.8 on COMPL. and +5.1 on FIM at $K = 8$); on the from-scratch cells it costs 2.8 on BASE and 4.1 on REWRITE while gaining 1.3 on UNSEEN, which is the trade-off a localizing read-out makes and the reason we report both. The line head pushes it further (+7.1 on COMPL., +1.8 on FIM, −6.5 to −15.0 on the from-scratch cells) and ends 5.0 points below the token ratio on average: it detects files with an interior AI segment rather than files as a whole.

5.2 LOCALIZATION VALUE AND BLOCK SIZE

The cell-level averages of Table 1 conflate how much of a file is AI with how its AI part is laid out, and only the latter bounds what localization can add. Stratifying the three mixed cells by the AI ratio ρ (Table 10, Appendix A.4) shows the bound directly: below $\rho = 0.2$ the token ratio and both block heads lie within 0.8 macro F1 of each other (50.4–51.2), while at $\rho \geq 0.8$ the K=8 head is 11.9 points above the token ratio (78.4 against 66.5). A supervised unit-level head buys little when the machine-written part of the file is small.

The block size follows the same logic: what matters is whether a block boundary can fall inside an AI segment. When the median segment is close to K, reducing K improves every bucket; when it is far below any usable K, most blocks are mixed and the larger K wins because each block carries more tokens and less feature noise. The three mixed cells of Table 22 instantiate the three regimes: on FIM (median segment near $K = 8$) and COMPL. (one segment far longer than either block) the smaller block wins (50.4 vs. 48.6 and 60.8 vs. 58.5 macro F1), as its interior blocks are pure and the finer grid places the boundary more precisely; on REFAC. (median two lines, below any usable K) the order reverses (62.9 vs. 63.8): no block isolates a segment, so tokens per block outweigh boundary precision. The line head is weaker still (54.8 file-level macro F1, line-level F1 48.0, Table 8): on REFAC. the signal comes from pooling, not localization. Tables 7 and 22 give both K per cell, and Table 9 per bucket (Appendices A.3, A.9, A.4).

Table 2: Ablations on the internal test set. Each row is a complete file-level system, and Δ is its paired-bootstrap difference from the panel's reference row (*: the 95% interval excludes zero). Further ablations: Appendix A.5.

	All cells		Mixed cells		TPR@1%FPR	
Variant	AUROC	Δ	AUROC	Δ	%	Δ
Token-level panel; reference: $s_{\text{ai-hum}}$, file mean						
$s_{\text{ai-hum}}$, file mean (ref.)	0.838	–	0.707	–	47.1	–
$s_{\text{ai-base}}$ only	0.773	-0.065^* [-0.074,-0.057]	0.646	-0.061^* [-0.073,-0.049]	18.9	-28.2^* [-30.5,-24.2]
$s_{\text{base-hum}}$ only	0.506	-0.331^* [-0.344,-0.318]	0.497	-0.210^* [-0.226,-0.193]	1.0	-46.0^* [-47.6,-42.7]
no ratio (AI-side NLL)	0.522	-0.316^* [-0.329,-0.301]	0.563	-0.144^* [-0.160,-0.128]	0.5	-46.6^* [-48.1,-43.4]
Token categories, parsed subset; reference: all tokens of the subset						
all tokens (ref.)	0.837	–	0.707	–	47.1	–
comments removed	0.802	-0.035^* [-0.041,-0.029]	0.688	-0.018^* [-0.027,-0.010]	21.4	-25.6^* [-28.1,-20.7]
code tokens only	0.744	-0.094^* [-0.103,-0.084]	0.645	-0.062^* [-0.075,-0.048]	6.4	-40.6^* [-43.3,-37.1]
comment tokens only	0.787	-0.051^* [-0.057,-0.044]	0.638	-0.069^* [-0.079,-0.058]	22.9	-24.2^* [-30.6,-17.0]
Block-level panel; reference: block head, K=16, BiGRU, logit mean						
block head K=16 (ref.)	0.853	–	0.749	–	48.4	–
$K = 8$	0.852	-0.001 [-0.004,$+0.003$]	0.761	$+0.013^*$ [$+0.007$,$+0.019$]	48.4	-0.0 [-2.9,$+4.0$]
logistic head, no BiGRU	0.845	-0.008^* [-0.013,-0.003]	0.750	$+0.001$ [-0.006,$+0.009$]	43.8	-4.6^* [-7.5,-0.1]
boosted-trees head	0.847	-0.006^* [-0.011,-0.000]	0.743	-0.006 [-0.015,$+0.004$]	41.3	-7.1^* [-10.4,-1.9]
no raw family	0.845	-0.008^* [-0.012,-0.004]	0.725	-0.024^* [-0.031,-0.016]	47.7	-0.7 [-3.4,$+3.5$]
no z-score family	0.843	-0.011^* [-0.014,-0.007]	0.737	-0.012^* [-0.017,-0.006]	44.1	-4.3^* [-7.8,-0.1]
no positional family	0.852	-0.001 [-0.002,$+0.001$]	0.746	-0.002 [-0.004,$+0.000$]	47.5	-0.9 [-2.7,$+2.1$]
no language family	0.856	$+0.003^*$ [$+0.001$,$+0.005$]	0.753	$+0.005^*$ [$+0.001$,$+0.008$]	47.9	-0.5 [-2.1,$+3.1$]
Adapter training data; reference: $s_{\text{ai-hum}}$, file mean						
10% of adapter data	0.783	-0.055^* [-0.063,-0.047]	0.670	-0.038^* [-0.051,-0.025]	27.9	-19.1^* [-21.7,-15.2]
Backbone family; reference: $s_{\text{ai-hum}}$, file mean						
deepseek-coder-1.3B backbone	0.826	-0.012^* [-0.017,-0.006]	0.690	-0.017^* [-0.026,-0.007]	43.2	-3.9^* [-6.4,-0.3]

5.3 Ablations

The ablations answer five questions: whether the ratio form is doing the work, which token categories carry it, whether any single feature family is load-bearing, how much of the performance depends on the amount of adapter training data, and how much of it depends on the backbone being Qwen2.5-Coder-1.5B. The components sit at two levels, so the panels are read against two references: the token-level panels against $s_{\text{ai-hum}}$ with the file-mean aggregation, the block-level panel against the $K = 16$ head with logit-mean aggregation. Each row removes or replaces exactly one component.

The token-level panel is the sharpest of the three. Neither half of the decomposition is sufficient: the AI-side deviation alone reaches 0.773 and the human-side deviation alone 0.506, i.e. chance, against 0.838 for the ratio. The asymmetry is the point: a term that cannot rank files at all is worth 0.065 AUROC and 28.2 TPR once subtracted, because what it removes is how predictable the text is: a normalizer, not a detector. Replacing the ratio by the AI-side adapter's likelihood (0.522) is no better, which rules out reading our score as "the AI-tuned model finds this file predictable".

In the block-level panel no feature family is individually load-bearing: the only two whose removal is a statistically detectable *cost* (-0.008 for raw, -0.011 for z-scores) are about an order of magnitude cheaper than either half of the ratio, the positional family is within noise on both AUROC columns (-0.001 and -0.002), and the language family is *negative* — removing it improves both AUROC columns slightly but significantly ($+0.003$ and $+0.005$) and leaves the tail unchanged within noise ($48.4 \rightarrow 47.9$). The positional result also holds where a shortcut would have to appear, on COMPL., whose AI segment is always at the tail: per-cell Δ (Table 11) costs at most 0.005 AUROC in any single cell, so the head does not take the positional shortcut.

The third reading concerns the operating point rather than any component. Rows that are nearly free in threshold-free terms are expensive at 1% FPR: removing comment tokens costs 0.035 AUROC but more than half of the TPR ($47.1 \rightarrow 21.4$), and a tenth of the adapter data costs 0.055 AUROC and again about 40% of the TPR. Scoring comments *alone* retains 0.787 AUROC, more than the code-token-only variant (0.744), which is itself 0.244 above chance, so comments are neither the mechanism nor a negligible part of it. A low-false-alarm deployment therefore depends on comment tokens and the full adapter training set — two components AUROC alone calls unimportant.

Table 3: Transfer (P1) and adaptation (P2) on AICD-Bench and DroidCollection (Orel et al., 2025). Group A trains nothing, so it has no P2 column and only its strongest row (DetectCodeGPT) is shown here; the full zero-shot set is in Table 24. ImBD's P2 uses length-matched pairs within each benchmark rather than the content-equivalent pairs its SPO stage assumes, since no external benchmark provides them (Appendix A.9). Aggregation and per-row exceptions are in Appendix A.9.

	Method	AICD-Bench T1		AICD-Bench T3		DroidCollection	
		P1	P2	P1	P2	P1	P2
		AUROC / TPR@1%FPR / macro F1 (%)					
A	DetectCodeGPT	0.703 / 6.5 / 49.7	–	0.666 / 0.2 / 40.2	–	0.669 / 3.0 / 35.1	–
B	CodeT5+ + MLP	0.471 / 1.3 / 44.8	0.629 / 11.6 / 53.5	0.462 / 0.7 / 40.7	0.927 / 43.3 / 73.5	0.521 / 0.8 / 32.7	0.982 / 81.5 / 89.7
B	CodeBERT	0.467 / 1.5 / 44.9	0.651 / 17.0 / 58.5	0.356 / 0.6 / 40.6	0.921 / 49.1 / 76.7	0.469 / 1.0 / 47.2	0.988 / 87.8 / 94.7
B	UniXcoder	0.489 / 1.9 / 45.3	0.618 / 12.6 / 55.0	0.374 / 0.7 / 40.7	0.929 / 35.8 / 68.9	0.499 / 1.5 / 50.8	0.991 / 90.1 / 96.2
B	*Same backbone*						
B	+ LoRA cls. head	0.654 / 5.8 / 49.1	0.636 / 20.8 / 61.3	0.550 / 0.7 / 40.7	0.967 / 64.6 / 84.6	**0.725** / 5.7 / 37.8	0.979 / 81.4 / 89.7
B	ImBD	0.606 / 5.2 / 48.5	0.549 / 14.9 / 56.9	0.564 / 2.4 / 42.4	*0.399* / 2.4 / 42.4	0.645 / 7.1 / 39.3	0.683 / 17.4 / 48.7
B	DroidDetect	0.535 / 1.9 / 45.4	0.386 / 0.1 / 43.6	0.469 / 0.2 / 40.2	0.891 / 42.0 / 72.7	0.532 / 1.1 / 51.4	0.988 / 86.1 / 96.0
C	Ours: token ratio, file mean	0.527 / 3.8 / 47.2	**0.829** / 11.9 / 54.4	0.484 / 2.3 / 42.4	0.951 / 29.9 / 65.1	0.525 / 3.9 / **51.8**	0.988 / 87.0 / 96.5
C	+ block head, $K = 8$	0.700 / 9.2 / 52.1	0.774 / 19.9 / 60.7	0.669 / 4.5 / 44.6	0.967 / 68.3 / 86.2	0.643 / 4.8 / 35.4	0.996 / 93.8 / 97.0
C	+ block head, $K = 16$	**0.773 / 15.9 / 57.7**	0.806 / 21.2 / 61.6	0.628 / 3.6 / 43.6	0.960 / **73.3** / **88.5**	0.603 / 3.3 / 33.3	0.996 / 93.5 / 97.0
C	+ line head, file mean	0.706 / 15.3 / 57.2	0.711 / **22.6** / **62.7**	**0.710** / **6.4** / **46.4**	**0.973** / 73.2 / 88.5	0.707 / **14.5** / 48.7	**0.996 / 94.0 / 97.0**

Swapping Qwen2.5-Coder-1.5B for deepseek-coder-1.3B (Guo et al., 2024) costs 0.012 AUROC and 3.9 TPR — a fifth of the data-quantity row, reversing no cell — although deepseek is the stronger language model, so predictive quality and ratio quality are distinct (Appendix A.5).

5.4 Is the signal just formatting?

Static formatting statistics separate human and machine code to a degree (Nirob et al., 2026), so we bound that explanation from both sides (Table 16, Appendix A.6): a whitespace-only logistic probe trained on the same corpus as the supervised baselines reaches 0.60 average AUROC, the static ceiling, while normalizing all formatting away — provably leaving the non-whitespace character sequence identical — costs the detector 3.5 AUROC points and still leaves the pure cells 25–30 points above their per-cell ceilings. The separation between methods in Table 1 is therefore not layout.

5.5 Robustness on real human code

A detector is only deployable if its false-positive rate on genuinely human-written code is low, measured on code that predates widespread model assistance rather than on the human side of a synthetic pair. Across $19{,}445$ files from repositories created before 2022, all of them negatives, at thresholds fixed on the development split and not re-tuned, all four read-outs stay between 0.55 and 0.81% against a nominal 1% (Table 18, Appendix A.7), so transporting the operating point from paired synthetic data to real repositories costs essentially nothing. Block- and line-level false-alarm rates on the same corpus are an order of magnitude higher (9.90% and 8.89% for blocks at $K = 8$ and $K = 16$, 17.33% for lines; Table 19): the file-level aggregation is what suppresses them, so any deployment that surfaces block- or line-level highlights has to be calibrated separately.

5.6 External benchmarks

Table 3 reports AICD-Bench and DroidCollection under the two protocols of Section 4.5; HCA is in Table 4. Appendix A.10 fixes what each benchmark contributes and states the DroidCollection class mapping. Where a benchmark is natively multi-class we additionally report the native multi-class number, so that it is never silently compared against a binary one.

On DroidCollection the P1 ranking is metric-dependent: the LoRA head has the best AUROC (0.725), our line head the best tail (TPR@1%FPR 14.5 against 7.1 for ImBD, the strongest baseline tail) and our token ratio the best macro F1 (51.8). Adaptation resolves the ranking — our line head leads on all three (0.996 / 94.0 / 97.0) — but not the tension between metrics: the token ratio beats both pretrained encoders on macro F1 once adapted (96.5 vs. 94.7 and 96.2) while having the weakest tail (87.0% against 87.8 and 90.1), so its errors sit where a low-false-alarm budget is spent.

Table 4: HybridCodeAuthorship (Patterson et al., 2026) under P1, at its native line granularity, our K-line blocks, and the file level. F1 here is plain, not macro. The weakest zero-shot statistics and the weakest fine-tuned encoder are omitted here; the full table is Table 25 in Appendix A.10.

	Method	HCA, P1 (zero adaptation)			
		Line	Block K=8	Block K=16	File
		AUROC / TPR@1%FPR / F1 (%)			
A	Fast-DetectGPT	0.495 / 1.2 / 2.2	0.568 / 1.8 / 3.5	0.568 / 1.9 / 3.7	0.499 / 0.0 / 0.0
	AIGCode Detector	0.646 / 1.4 / 2.7	0.513 / 2.2 / 4.1	0.501 / 1.7 / 3.2	0.496 / 1.4 / 16.7
B	CodeBERT	**0.666** / 6.1 / 11.4	0.577 / 4.8 / 9.0	0.589 / 5.4 / 10.1	0.594 / 3.3 / 6.3
	UniXcoder	0.581 / 3.6 / 6.8	0.562 / 4.8 / 9.0	0.570 / 5.2 / 9.8	0.580 / 2.4 / 4.6
	Same backbone + LoRA cls. head	*0.337* / 0.6 / 1.1	0.532 / 1.3 / 2.5	0.534 / 1.4 / 2.7	0.476 / 1.2 / 2.4
	ImBD	0.536 / 1.3 / 2.5	0.531 / 2.6 / 5.0	0.553 / 3.0 / 5.7	0.472 / 2.2 / 4.3
	DroidDetect	0.586 / 2.6 / 5.0	0.682 / 6.6 / 12.2	**0.685** / 9.3 / 16.8	0.712 / 11.3 / 26.7
C	Ours: token ratio, line mean	0.528 / 0.9 / 53.5	0.637 / 6.9 / 49.5	0.628 / 9.2 / **52.1**	0.638 / 12.8 / 22.7
	+ block head, $K = 8$	–	**0.685 / 14.0 / 51.5**	–	0.690 / 15.3 / 26.5
	+ block head, $K = 16$	–	–	0.627 / **11.0** / 50.4	0.667 / 14.4 / 25.2
	+ line head, run mean	0.589 / **10.6** / **62.5**	–	–	**0.786 / 19.1 / 32.0**

The line head's edge under transfer — the strongest of our read-outs on T3 P1 and on DroidCollection's P1 — does not survive adaptation: on T1 P2 it is the weakest of our own four rows (0.711 against 0.829 for the token ratio), although still above every Group B row there. The asymmetry is expected rather than incidental: T1, T3 and DroidCollection annotate at the file level, so every line of a positive file carries the same label and the line head has no localization supervision to exploit — it degenerates to a file-level classifier with a mean-pooled loss, and whatever edge it has under transfer comes from the sequence model rather than from line-level evidence.

On HCA the line head is the strongest of our read-outs at both granularities it covers: 62.5 line-level $F1$, and 0.786 AUROC at the file level, the best AUROC anywhere in the table (0.690 / 0.667 for the block heads, 0.712 for the strongest baseline; Table 4). The margin is at the file level: at the line granularity every method stays near chance, ours at 0.589. That fits the segment-length analysis of Section 5.2: HCA's median span is two lines and its median AI fraction is 0.21, so per-line evidence is thinnest exactly where only the file-level aggregate carries weight.

6 Limitations

Our mixed files use few writer models and prompts, and the specification-based construction strips machine-side dead code and license headers, so the numbers are an upper bound, not a deployment estimate. Training sees only long AI segments (median 20 lines), so the scattered regime is measured but never trained on, and the positional family, bounded at -0.001 AUROC here, is not ruled out elsewhere. Byte-identical low-entropy pairs put the same text on both sides as known label noise.

7 Conclusion

Detecting AI-involved code is a different problem from detecting AI-written code, and the difference is not a matter of degree. Where a generated span sits inside surviving human code, the supervised file-level control that reaches 71.6–79.3 macro F1 on whole machine-written files stays 3.6 and 1.5 points above the all-negative floor, against 11.4 and 5.5 for the token-level likelihood ratio (Table 1): a windowed head averages a short inserted span away, while the ratio between an AI-side and a human-side adapter keeps it addressable and the line and block heads read it where it sits. The mechanisms are therefore complementary rather than competing.

What bounds a localizing read-out is the size of the machine-written part relative to the prediction unit, not the choice of unit: below $\rho = 0.2$ the token ratio and both block heads lie within 0.8 macro F1 of each other, and once the median AI segment falls below any usable block — two lines on REFAC. — the ordering of K reverses and the line head drops to 54.8 file-level macro F1 at 48.0 line-level F1, so what remains is pooling rather than attribution. The transfer–adaptation gap carries the same message one level up: most of what separates the classes on AICD-Bench and DroidCollection is specific to how each corpus was generated rather than a property of machine-written code that survives a change of protocol.

AI USE STATEMENT

LLMs were used in this work in the following limited capacities.

1. **Dataset construction.** The AI-generated code portion of our internal dataset was constructed by calling LLM APIs. The training, validation and main test splits use four writer models — Claude Opus 5, DeepSeek-V4-Flash, GPT-5.6 and GLM-5.2; the held-out-model generalization test additionally uses Qwen3-Coder-Plus, Doubao-Seed-2.1-Pro, Kimi-K2.6 and MiniMax-M2.7.
2. **Related-work search.** We used LLMs to assist in searching for and surveying related work.
3. **Experiment implementation.** We used a coding agent to implement the experiment scripts according to our pre-specified experimental plan.
4. **Translation.** The paper was initially drafted in Chinese and we used LLMs to translate it into English.
5. **Writing.** All sections of this paper were initially drafted and completed by the human authors, who established all core ideas, analyses and conclusions. LLMs were then used to refine and polish these pre-written drafts.

They were not used for proposing or refining hypotheses, for designing or giving feedback on methodology or experiments, for formulating mathematical claims or proofs, for cleaning or reformatting data, for qualitative or thematic data analysis, or for interpreting results; no other task on the required-disclosure list applies to this work. We have reviewed all AI-assisted work: the experiment code produced with the coding agent was verified and tested for correctness by one of the authors. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Detectors of this kind are deployed in assessment and code-provenance settings where a false positive is costly, which is why every headline number in this paper is reported at a low-false-positive operating point and why we measure the false-positive rate on genuinely human code from repositories that predate widespread model assistance rather than on the human side of a synthetic pair. Two consequences of our measurements bear directly on responsible use. First, the false-alarm rate is language-dependent, and the worst per-language rate we observe is an order of magnitude above the aggregate, so a single global threshold disadvantages authors writing in the less-represented languages. Second, block- and line-level highlights carry much higher false-alarm rates than the file-level verdict, so surfacing them to a human reviewer without separate calibration would present unreliable evidence as if it were localized proof. We release no model that scores individual authors, and the training corpus is built from public repositories under their original licenses.

REPRODUCIBILITY STATEMENT

The dataset construction scripts, exact split definitions, frozen backbone and adapter configuration, scoring pipeline, head-training settings, evaluation protocols, table conventions, aggregation choices, per-table exceptions, random seeds, and run-to-run variation are documented in Sections 4.1– 4.4 and Appendices A.8–A.9. The complete code pipeline — dataset construction, scoring, head and aggregator training, and the evaluation scripts behind every table — is anonymized and submitted as supplementary material with this PDF, and will be released as a permanent versioned repository after acceptance. The internal corpus is not released during review; after publication, it will be made available through an application process, with access requested by email and governed by the original licenses of the source repositories.

REFERENCES

Guangsheng Bao, Yanbin Zhao, Zhiyang Teng, Linyi Yang, and Yue Zhang. Fast-DetectGPT: Efficient zero-shot detection of machine-generated text via conditional probability curvature. In *International Conference on Learning Representations (ICLR)*, 2024.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of EMNLP*, 2020.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander M. Rush. GLTR: Statistical detection and visualization of generated text. In *Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*, 2019.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. UniXcoder: Unified cross-modal pre-training for code representation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

Abhimanyu Hans, Avi Schwarzschild, Valeriia Cherepanova, Hamid Kazemi, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Spotting LLMs with Binoculars: Zero-shot detection of machine-generated text. In *International Conference on Machine Learning (ICML)*, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. DetectGPT: Zero-shot machine-generated text detection using probability curvature. In *International Conference on Machine Learning (ICML)*, 2023.

Syed Mehedi Hasan Nirob, Shamim Ehsan, Moqsadur Rahman, and Summit Haque. Whitespaces don't lie: Feature-driven and embedding-based approaches for detecting machine-generated code. *arXiv preprint arXiv:2601.19264*, 2026.

Yuling Shi, Hongyu Zhang, Chengcheng Wan, and Xiaodong Gu. Between lines of code: Unraveling the distinct patterns of machine and human programmers. In *International Conference on Software Engineering (ICSE)*, 2025.

Jiaqi Chen, Xiaoye Zhu, Tianyang Liu, Ying Chen, Xinhui Chen, Yiwen Yuan, Chak Tou Leong, Zuchao Li, Long Tang, Lei Zhang, Chenyu Yan, Guanghao Mei, Jie Zhang, and Lefei Zhang. Imitate before detect: Aligning machine stylistic preference for machine-revised text detection. In *AAAI Conference on Artificial Intelligence*, 2025.

Jinyan Su, Terry Yue Zhuo, Di Wang, and Preslav Nakov. DetectLLM: Leveraging log rank information for zero-shot detection of machine-generated text. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. CodeT5+: Open code large language models for code understanding and generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Zhenyu Xu and Victor S. Sheng. Detecting AI-generated code assignments using perplexity of large language models. In *AAAI Conference on Artificial Intelligence*, volume 38, pages 23155–23162, 2024.

Daniil Orel, Indraneil Paul, Iryna Gurevych, and Preslav Nakov. Droid: A resource suite for AI-generated code detection. *arXiv preprint arXiv:2507.10583*, 2025.

Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. AICD Bench: A challenging benchmark for AI-generated code detection. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Volume 1: Long Papers*, pages 6913–6938, 2026.

Luke S. Patterson, Li Wang, and Adam Faulkner. HybridCodeAuthorship: A benchmark dataset for line-level code authorship detection. In *Proceedings of the Language Resources and Evaluation Conference (LREC)*, pages 1520–1532, 2026.

Phuong T. Nguyen, Juri Di Rocco, Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, and Massimiliano Di Penta. GPTSniffer: A CodeBERT-based classifier to detect source code written by ChatGPT. *Journal of Systems and Software*, 214:112059, 2024.

Xiaodan Xu, Chao Ni, Xinrong Guo, Shaoxuan Liu, Xiaoya Wang, Kui Liu, and Xiaohu Yang. Distinguishing LLM-generated from human-written code by contrastive learning. *arXiv preprint arXiv:2411.04704*, 2024.

Oseremen Joy Idialu, Noble Saji Mathews, Rungroj Maipradit, Joanne M. Atlee, and Meiyappan Nagappan. Whodunit: Classifying code as human authored or GPT-4 generated — a case study on CodeChef problems. In *International Conference on Mining Software Repositories (MSR)*, 2024.

Sufiyan Ahmed Bukhari, Benjamin Tan, and Lorenzo De Carli. Distinguishing AI- and human-generated code: A case study. In *Proceedings of the 2023 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '23)*, 2023.

Pengyu Wang, Linyang Li, Ke Ren, Botian Jiang, Dong Zhang, and Xipeng Qiu. SeqXGPT: Sentence-level AI-generated text detection. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Osama Mohammed Afzal, Tarek Mahmoud, Giovanni Puccetti, Thomas Arnold, Chenxi Whitehouse, Alham Fikri Aji, Nizar Habash, Iryna Gurevych, and Preslav Nakov. SemEval-2024 Task 8: Multigenerator, multidomain, and multilingual black-box machine-generated text detection. In *International Workshop on Semantic Evaluation (SemEval)*, 2024.

Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Osama Mohammed Afzal, Tarek Mahmoud, Giovanni Puccetti, Alham Fikri Aji, Nizar Habash, Iryna Gurevych, and Preslav Nakov. M4GT-Bench: Evaluation benchmark for black-box machine-generated text detection. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Liam Dugan, Daphne Ippolito, Arun Kirubarajan, and Chris Callison-Burch. RoFT: A tool for evaluating human detection of machine-generated text. In *Conference on Empirical Methods in Natural Language Processing (EMNLP), System Demonstrations*, 2020.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. DExperts: Decoding-time controlled text generation with experts and anti-experts. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

Alisa Liu, Xiaochuang Han, Yizhong Wang, Yulia Tsvetkov, Yejin Choi, and Noah A. Smith. Tuning language models by proxy. In *Conference on Language Modeling (COLM)*, 2024.

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. DeepSeek-Coder: When the large language model meets programming — the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Rebecca Killick, Paul Fearnhead, and Idris A. Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, 2020.

A Appendix

Table 5 maps each part of this appendix to the main-text section it resolves. Two parts are prerequisites for the rest rather than extensions of a single result: Appendix A.8 fixes the configuration behind every number we report, and Appendix A.9 fixes the conventions every table inherits together with its per-table exceptions. Appendix A.5 collects ablations and one negative result that no main-text claim rests on. Everything else supplies the per-cell, per-bucket or per-language resolution of a result the main text states in aggregate.

Table 5: Guide to the appendix. The last column is the main-text section whose result the appendix resolves, or which defers its material here.

Appendix	Contents	Main text
A.1 Extended related work	zero-shot likelihood statistics, supervised code detectors, two-model contrasts, spans and boundaries in text, mixed and fine-grained authorship resources	1
A.2 Accuracy and F1 at the transported threshold	why neither metric appears in the main text; accuracy per test cell and per synthesis form against the degenerate baseline	4.4
A.3 Block-level metrics	block and line metrics of both heads, per test cell, at both K	5.2
A.4 Localization diagnostics	localization value by AI segment length and by AI ratio ρ	5.2
A.5 Additional ablations	feature families per cell, the aggregation sweep, the line-head read-out, the block-supervision corpus, syntax-aligned blocks, the backbone swap	3.2, 3.3, 5.3
A.6 Formatting controls	whitespace and length ceilings; the two levels of formatting normalization	5.4
A.7 False positives on real repository code	false-alarm rates per language, under the deployment and the accuracy-maximising threshold	5.5, Ethics
A.8 Implementation details	dataset splits, backbone and adapter configuration, scoring passes, head inputs and training	3.1, 3.2, 4.1, Reprod.
A.9 Table conventions and protocol details	the conventions every table inherits, per-table exceptions, the main comparison with all three metrics per cell	4.3, 4.4, 5.1
A.10 External benchmarks	benchmark composition and class mapping, full zero-shot and HybridCodeAuthorship tables	4.5, 5.6

A.1 Extended related work

This section is the related-work discussion deferred from the main text.

Zero-shot likelihood detectors. Zero-shot detectors threshold a likelihood statistic computed under a reference model: perturbation curvature, estimated by mask-filling (Mitchell et al., 2023) or in analytic conditional-probability form (Bao et al., 2024); log-rank (Gehrmann et al., 2019) and its ratios LRR and NPR (Su et al., 2023); and cross-perplexity between a performer and an observer model, which cancels much of the prompt-difficulty confound (Hans et al., 2024). They need no labels and transfer across generators, and they all rest on the premise that machine text is more predictable than human text under a reference model.

Supervised detectors for machine-generated code. DetectCodeGPT is the code-side perturbation detector, inserting newlines and whitespace instead of filling masked natural-language spans, on the observation that machine code is more regular in exactly those positions (Shi et al., 2025). Most supervised code detectors instead train a discriminator on labelled source: CodeBERT- and CodeGPT-based classifiers (Nguyen et al., 2024; Xu et al., 2024), detectors built on surface-form, whitespace or embedding features (Nirob et al., 2026), and function-level detectors for Python (Idialu et al., 2024) and C programs (Bukhari et al., 2023); all of them assume a labelled pool in the language and style of the code under test. ImBD aligns a proxy model to machine stylistic preference by direct preference optimization and then reads a curvature statistic under the aligned model (Chen et al., 2025), the strongest recent supervised-scoring competitor for machine-revised text. Our own supervised controls are described in Section 4.3.

Contrasting two models. A difference of two models' log-probabilities over a shared backbone is the score form used in decoding-time control — contrastive decoding (Li et al., 2023), DExperts (Liu et al., 2021) and proxy tuning (Liu et al., 2024) — where the two sides are experts steering generation rather than detectors. What we take from that line is the form of the score; what we add is its role as a per-token detection statistic and the sequence heads that read it.

Spans and boundaries in text. Below the document level, the nearest formulations come from machine-generated text detection. SeqXGPT labels machine-written sentences with a token-level classifier (Wang et al., 2023*b*); SemEval-2024 Task 8 Subtask C asks where the boundary between human and machine spans falls inside a mixed document (Wang et al., 2024*a*); M4GT-Bench organizes datasets and domains around the same mixed-text setting (Wang et al., 2024*b*), and RoFT studies where human readers place such boundaries (Dugan et al., 2020). The task is the same here and the structural assumptions are not. Mixed text is generated forward from one prompt, so authorship changes at a point in the sequence and the natural procedures are change-point detection or token tagging. A mixed file is assembled by local edits against a surrounding context, with fill-in-the-middle insertions and whole-block rewrites, so AI involvement is block-aligned rather than a single cut. That difference is what our block and line heads encode, and it is why we report attribution at those two granularities instead of a boundary position.

Mixed and fine-grained authorship. Most resources for code authorship operate at the file level: DroidCollection is a multi-generator, multi-domain corpus annotated over four authorship classes, which also ships the fine-tuned encoder detector DroidDetect (Orel et al., 2025), and AICD-Bench stays at the file level and targets out-of-distribution robustness through its own tracks and splits (Orel et al., 2026). HybridCodeAuthorship is the exception: it provides per-line labels for human–AI co-written files with no training split (Patterson et al., 2026), with a median AI segment of two lines.

A.2 Accuracy and F1 at the transported threshold

Neither accuracy nor F1 appears in the main text, because the threshold both require does not transport. The development split contains 8.9% mixed files and the test set 32.8%, so a threshold that maximizes accuracy on development sits in the wrong place on test, and the test set is additionally imbalanced at $4.63{:}1$ positives to negatives. Under those conditions the degenerate all-positive classifier reaches 0.903 F1 and 0.822 accuracy on the internal test set, and all three of our read-outs fall below both: 0.795, 0.810 and 0.797 F1, and 0.717, 0.734 and 0.719 accuracy, for the token ratio and the two block heads. This is the reason the main comparison uses AUROC and TPR at a fixed FPR: both are computable without a transported threshold, and neither can be won by a degenerate predictor.

Table 6 resolves that same threshold per cell and per synthesis form, and it has to be read across cells rather than down a column. Every cell except HUMAN contains positives only, so accuracy there is recall and the all-positive baseline is 1.000 by construction; the HUMAN column is 1 minus the false-alarm rate at the same threshold and is the only column that constrains the threshold from the other side. The per-form panel is where the degenerate baseline carries information, because its *pure* column pools the pure-AI files with the human pool (0.736 all-positive accuracy): the token ratio clears that floor at 0.910, while all three mixed forms sit far below the 1.000 their own column demands — 0.127 for the token ratio on fill-in-the-middle, against 0.275 for the $K{=}8$ head, which

Table 6: Accuracy at the transported accuracy-maximising threshold, per test cell (upper panel) and per synthesis form (lower panel). Each read-out uses its own threshold, frozen on the development split by maximizing development accuracy and applied unchanged to test; the three values are those of the lower panel of Table 18. *all-pos.* is the degenerate all-positive accuracy, i.e. the floor of that column. Every cell except HUMAN holds positives only, so its accuracy equals recall and its floor is 1.000; the HUMAN column is 1 minus the false-alarm rate. In the lower panel *pure* is the 9,619 files with no mixing — the three from-scratch cells plus the human pool — which is why its floor is 0.736 rather than 1.000. The file-level accuracies quoted in the text are these per-cell values weighted by cell size.

Read-out	threshold	BASE	UNSEEN	REWRITE	COMPL.	REFAC.	FIM	HUMAN
Token ratio, file mean	0.00081	0.897	0.896	0.893	0.263	0.506	0.127	0.949
+ block head, K=8	−1.434	0.844	0.911	0.804	0.490	0.527	0.275	0.923
+ block head, K=16	−2.107	0.846	0.909	0.803	0.422	0.487	0.212	0.943
all-pos. floor	–	1.000	1.000	1.000	1.000	1.000	1.000	0.000

Read-out	*pure* (n = 9,619)	continuation (1,496)	refactor diff (1,899)	fill-in-middle (1,300)
Token ratio, file mean	0.910	0.263	0.506	0.127
+ block head, K=8	0.874	0.490	0.527	0.275
+ block head, K=16	0.879	0.422	0.487	0.212
all-pos. floor	0.736	1.000	1.000	1.000

is the same ordering the AUROCs of Table 1 give and the reason a localizing read-out is worth its cost on the from-scratch cells.

A.3 BLOCK-LEVEL METRICS

Block-level AUROC and F1 for each K. All block labels follow Section 2: a block is positive if it contains at least one AI line. On the internal test set this makes 72.0% of blocks positive at $K = 8$ and 74.6% at $K = 16$, and the block head reaches 0.874 AUROC at $K = 8$ and 0.882 at $K = 16$ over all blocks, against 0.906 and 0.930 when the metric is restricted to blocks that are pure in content. At the development-frozen threshold its F1 is 82.8 ($K = 8$) and 80.4 ($K = 16$), which is below the all-positive floors of 83.7 and 85.5: at a positive rate above 0.7, F1 rewards the degenerate classifier, and AUROC is the comparable reading at this granularity.

Table 7 resolves both K by cell, and it separates two things the pooled numbers above conflate. Within a cell the smaller block is at least as good on all three mixed cells (0.912, 0.648 and 0.650 at $K = 8$ against 0.897, 0.646 and 0.594 at $K = 16$), which is the ordering the lower panel of Table 9 also shows; pooled over all blocks the ordering reverses (0.874 against 0.882), because pooling mixes blocks from different cells and the easy separation between the all-positive blocks of the pure cells and the all-negative blocks of the HUMAN cell dominates the ranking. Block size therefore has to be chosen inside a cell, not on the pooled number.

The line head is reported per cell in Table 8, since its line-level behaviour is what bounds a localize-first pipeline. Line-level AUROC is defined only on the three mixed cells: in the pure cells every line is AI and in the HUMAN cell none is. The pure rows therefore carry a recall but no ranking metric, and the HUMAN row carries neither — with no positive line its recall and $F1$ are undefined as well, and it appears only to fix the negative pool.

The line-level signal is regime-dependent rather than uniformly weak: 0.927 on COMPL., where the AI part is one long tail segment, against 0.599 on REFAC., whose median segment is two lines, and 0.695 on FIM. On REFAC. the transported line-level $F1$ is 48.0; on COMPL. it reaches 85.2.

A.4 LOCALIZATION DIAGNOSTICS

Table 9 is the evidence behind Section 5.2. Its upper panel separates the aggregation question from the difficulty question. *Token ratio, file* is the deployed read-out; *oracle* restricts the positive statistic to the tokens inside the true AI spans while holding the negative pool fixed, so the step from *file* to *oracle* measures what the human majority costs the file-level average. It is a positive-side ablation

Table 7: Block-level metrics of the block head per test cell, at both K. Block labels are those of Section 2: positive if the block contains at least one AI line. *pos* is the block positive rate; AUROC ranks blocks inside the cell and is undefined where every block carries the same label; $F1$ and *recall* use the block-level threshold frozen on the development split (0.3214 at $K = 8$, 0.2271 at $K = 16$) and transported unchanged. The HUMAN cell has no positive block, so its $F1$ and recall are undefined; it fixes the negative pool of the pooled numbers quoted above.

Cell	blocks	pos	AUROC	$F1$	recall
$K = 8$, development threshold 0.3214					
BASE	31,369	1.000	–	89.6	81.2
UNSEEN	35,225	1.000	–	92.9	86.7
REWRITE	23,777	1.000	–	86.6	76.4
COMPL.	19,131	0.570	0.912	80.0	71.3
REFAC.	22,010	0.655	0.648	62.6	53.0
FIM	14,379	0.639	0.650	45.8	31.9
HUMAN	27,607	0.000	–	–	–
$K = 16$, development threshold 0.2271					
BASE	16,328	1.000	–	89.2	80.5
UNSEEN	18,249	1.000	–	92.7	86.4
REWRITE	12,388	1.000	–	84.5	73.2
COMPL.	9,952	0.628	0.897	71.0	56.8
REFAC.	11,483	0.762	0.646	58.2	44.0
FIM	7,516	0.728	0.594	34.5	21.5
HUMAN	14,460	0.000	–	–	–

Table 8: Line head, per cell. Line labels are those of Section 2. *Line AUROC* ranks lines within the cell; it is undefined where all lines carry the same label (BASE, UNSEEN, REWRITE, HUMAN). $F1$ uses the threshold -0.272, selected on the development splits by line-level $F1$ and transported unchanged. Recall is at the same threshold. A dash means undefined: the HUMAN cell contains no positive line, so its recall and $F1$ have no value, and the quantity that would take their place — a line-level false-alarm rate — is not reported at this granularity.

Cell	files	lines	AI-line rate	Line AUROC	Line $F1$	Line recall
BASE	2,541	237,027	1.000	–	91.2	83.8
UNSEEN	2,541	267,667	1.000	–	93.4	87.6
REWRITE	1,996	178,961	1.000	–	86.2	75.7
COMPL.	1,496	143,999	0.519	0.927	85.2	90.7
REFAC.	1,899	165,466	0.334	0.599	48.0	61.2
FIM	1,300	107,925	0.453	0.695	58.7	55.1
HUMAN	2,541	206,146	0.000	–	–	–

and not a like-for-like comparison — a sub-mean over a ground-truth-selected token subset against a whole-file mean — so we do not read it as a performance number. *oracle-pair* additionally replaces the negative with the same line range of the file's paired human original, which makes both sides the same statistic and controls for content difficulty, and that is the only variant we read as an upper bound on localization value: a short-segment bucket is itself composed of low-entropy files, so *oracle* alone overstates the signal. The *length-only* probe is reported for the same reason at the file level. The lower panel reads the block head at its own granularity and at the file level from the mean of the same logits, so that the two effects of the head (better attribution and better file-level pooling) can be told apart.

Stratification by AI ratio. Table 10 reads the four Group C rows of Table 1 against ρ rather than against segment length, on the three mixed cells pooled. All four rise monotonically with ρ, and the unit-level heads separate from the token ratio only above $\rho \approx 0.4$: in the lowest bucket the four read-outs lie within 0.022 AUROC of each other (0.642–0.664), while at $\rho \geq 0.8$ the K=8 head is 0.051 AUROC and 21.0 TPR points above the token ratio. The stratification is therefore a second statement of the bound of Section 5.2: a supervised unit-level head buys little when the machine-written part of the file is small, whether "small" is measured as a short segment or as a low ratio.

Table 9: Localization value and block size by segment length, on the two interior-segment cells COMPL. and FIM (pooled). A file's bucket is the median length, in non-blank lines, of its AI segments. *Upper panel:* the token-level ratio aggregated over the whole file, over the true AI spans (*oracle*), and against the same line range of the paired human original (*oracle-pair*, the strictest control and the only variant we read as an upper bound on localization value); *length-only* is a logistic probe on line and character count, included as a difficulty control. *Lower panel:* the block head read at block level, against the block labels of Section 2, and at file level, by the mean of the same logits. All entries are threshold-free AUROC. Segment merging and the negative pools are defined in Appendix A.9.

Variant	≤ 4	5–9	10–24	≥ 25	All
#positive files	439	449	756	1,152	2,796
Token-level aggregation					
Token ratio, file	0.510	0.561	0.630	0.775	0.660
Token ratio, oracle	0.650	0.738	0.840	0.920	0.827
Token ratio, oracle-pair	0.560	0.654	0.771	0.888	0.758
Length-only probe	0.453	0.497	0.543	0.680	0.578
Block head (choosing K)					
Block AUROC, $K = 8$	0.538	0.615	0.718	0.900	0.803
Block AUROC, $K = 16$	0.520	0.606	0.701	0.883	0.776
File AUROC, $K = 8$	0.549	0.596	0.722	0.889	0.743
File AUROC, $K = 16$	0.531	0.580	0.691	0.863	0.719

Table 10: File-level detection stratified by the AI ratio ρ, on the three mixed-authorship cells COMPL., REFAC. and FIM (pooled). A file's column is its ρ (Section 2) and the boundaries are the five construction buckets of Section 3.4, except that the lowest bucket extends down to 0 and the highest includes $\rho = 1$: REFAC. is produced by refactoring diffs whose coverage is not solved backwards from a target ρ, so 177 of its files fall below 0.05 and 15 at or above 0.95. Each entry is AUROC / TPR@1%FPR / macro F1 (%). Metrics, negative pool, chance levels and bolding follow Section 4.4. Bold marks the highest AUROC of a column. The positive set is the files of that column; the negative pool is the shared HUMAN cell ($2{,}541$ files) in every column, and the threshold behind both operating-point metrics is the single 99th percentile of its scores, so TPR@1%FPR is comparable across columns. Chance is AUROC 0.500 and TPR 1.0%. Macro F1 is *not* comparable across columns, because the positive count changes while the negative pool does not; its all-negative reference, the chance level at this operating point, is given as a row.

Method	< 0.2	0.2–0.4	0.4–0.6	0.6–0.8	≥ 0.8	All
#positive files	1,298	1,105	914	801	577	4,695
all-negative macro F1	39.8	41.1	42.4	43.2	44.9	26.0
Token ratio, file mean	0.642/11.2/50.4	0.702/17.8/57.0	0.716/17.6/57.9	0.778/23.3/62.8	0.858/27.0/66.5	0.720/18.0/43.5
+ block head, K=8	**0.664**/11.3/50.5	**0.731**/18.6/57.6	0.796/22.3/61.4	**0.858**/35.0/70.4	**0.909**/48.0/78.4	**0.769**/23.7/48.2
+ block head, K=16	0.653/12.1/51.2	0.727/19.9/58.6	0.786/20.4/60.0	0.842/32.0/68.5	0.901/43.2/75.9	0.759/22.7/47.4
+ line head, file mean	0.646/8.3/47.8	0.713/12.6/52.7	**0.799**/14.9/55.7	0.843/23.1/62.6	0.900/38.3/73.3	0.756/16.8/42.4

A.5 ADDITIONAL ABLATIONS

These tables extend Table 2 to the components whose ablation does not change the main conclusions: the per-cell reading of the feature families, the aggregation sweep, the line-head read-out, the construction of the block supervision, and one negative result on block partitioning. Unless a caption says otherwise the protocol is that of Table 2: file-level scores, the shared HUMAN cell ($2{,}541$ files) as the negative pool, Δ as a paired bootstrap difference against the panel's reference row ($1{,}000$ replicates, 95% interval, threshold recomputed inside each replicate), and * for an interval that excludes zero. Chance is AUROC 0.500 and TPR 1.0%.

Feature families per cell. Table 11 resolves the four family rows of Table 2 by cell, which is where a shortcut to the construction prior would have to show up.

Neither the positional nor the language family moves any cell by more than 0.007, and on COMPL., the cell whose segment position is perfectly predictable, removing the positional family costs 0.005. The raw and z-score families are complementary rather than redundant: removing the raw family

Table 11: Feature-family ablations of the $K{=}16$ block head, per test cell. The first row is the full head's AUROC on each cell; the remaining rows are *paired differences* of a head retrained without that family, computed on that cell alone (*: 95% bootstrap interval excludes zero). Each column scores that cell's positive files against the shared HUMAN cell. The prior that the positional family could exploit is cell-specific: the AI segment is always at the tail in COMPL., always in the interior in FIM, and scattered over the whole file in REFAC.

	BASE	UNSEEN	REWRITE	COMPL.	REFAC.	FIM
Block head $K{=}16$ (AUROC)	0.954	0.974	0.944	0.789	0.819	0.638
Δ no raw family	$+0.007^*$	-0.007^*	$+0.023^*$	-0.036^*	-0.004	-0.031^*
Δ no z-score family	-0.008^*	-0.010^*	-0.010^*	-0.014^*	-0.026^*	$+0.004$
Δ no positional family	$+0.001$	-0.001	$+0.001$	-0.005^*	-0.001	$+0.001$
Δ no language family	$+0.000$	-0.000	$+0.004^*$	$+0.006^*$	$+0.001$	$+0.007^*$

Table 12: Aggregation sweep. Upper block: the $K{=}16$ block head read out by the six aggregations of its per-block probabilities defined above. Lower block: the token-level ratio under the four aggregations of Section 3.3. All Δ are paired differences against the single reference row, so the lower block's Δ contains both the change of read-out and the change of aggregation. Columns, protocol and * are as in Table 2.

	All cells		Mixed cells		TPR@1%FPR	
Aggregation	AUROC	Δ	AUROC	Δ	%	Δ
Block head, $K{=}16$						
logit mean (ref.)	0.853	–	0.749	–	48.4	–
mean	0.853	$-0.001\,[-0.002,+0.001]$	0.752	$+0.004^*\,[+0.001,+0.006]$	47.9	$-0.5\,[-2.2,+1.8]$
top-25% mean	0.850	$-0.003^*\,[-0.005,-0.000]$	0.754	$+0.006^*\,[+0.001,+0.010]$	45.2	$-3.2\,[-5.3,+0.1]$
top-10% mean	0.849	$-0.004^*\,[-0.007,-0.001]$	0.753	$+0.005\,[-0.000,+0.010]$	44.2	$-4.2^*\,[-5.9,-0.2]$
max	0.848	$-0.005^*\,[-0.008,-0.002]$	0.752	$+0.004\,[-0.001,+0.009]$	44.1	$-4.3^*\,[-6.1,-0.3]$
best adjacent pair	0.846	$-0.007^*\,[-0.010,-0.003]$	0.754	$+0.005\,[-0.001,+0.012]$	42.8	$-5.6^*\,[-7.5,-1.8]$
Token ratio $s_{\text{ai-hum}}$						
file mean	0.838	$-0.015^*\,[-0.021,-0.010]$	0.707	$-0.041^*\,[-0.051,-0.032]$	47.1	$-1.3\,[-4.9,+3.2]$
sliding-window max	0.827	$-0.026^*\,[-0.034,-0.018]$	0.715	$-0.034^*\,[-0.046,-0.021]$	41.7	$-6.7^*\,[-10.2,-2.0]$
change-point tail	0.820	$-0.033^*\,[-0.040,-0.025]$	0.704	$-0.044^*\,[-0.055,-0.033]$	22.4	$-26.0^*\,[-31.0,-16.5]$
best segment	0.838	$-0.016^*\,[-0.021,-0.010]$	0.707	$-0.041^*\,[-0.051,-0.031]$	47.0	$-1.4\,[-4.8,+3.4]$

improves BASE and REWRITE (up to $+0.023$ on REWRITE) and costs 0.007 on UNSEEN, while costing the most on the two interior-segment cells (-0.036 on COMPL., -0.031 on FIM), so what it contributes is specific to reading a boundary inside a file; removing the within-file z-scores costs 0.008–0.026 on five of the six cells instead.

Aggregation. Table 12 sweeps the six file-level aggregations of the block probabilities and the four token-level aggregations of Section 3.3 against one common reference. The ten read-outs are implemented as follows. Over the per-block probabilities $\hat{z}_b$ of a file's block set $\mathcal{B}$: *logit mean* averages $\sigma^{-1}(\hat{z}_b)$ after clipping the probabilities away from 0 and 1 (Equation 2); *mean* averages $\hat{z}_b$ in probability space instead; *top*-25% and *top*-10% *mean* sort the blocks by $\hat{z}_b$ and average the highest $\lceil 0.25|\mathcal{B}| \rceil$ and $\lceil 0.10|\mathcal{B}| \rceil$ of them, at least one block in either case; *max* takes the single highest $\hat{z}_b$, which carries a length bias the two top-share means do not, since a file with more blocks has a systematically higher extreme; and *best adjacent pair* is $\max_b \min(\hat{z}_b, \hat{z}_{b+1})$, a "two blocks in a row are high" statistic that encodes the prior that an AI segment covers more than one block and is less sensitive to a single noisy block than the max. Over the per-token ratio s_t: *file mean* is Equation 2; *sliding-window max* takes the largest within-window mean of s_t over 128-token windows at stride 32, falling back to the file mean for files shorter than one window; *change-point tail* places a single cut with a CUSUM statistic on s_t and averages the tokens after it, keeping the file mean when fewer than 8 tokens remain; and *best segment* splits the file by recursive binary segmentation — at most 8 cuts, each segment at least 16 tokens, and a split accepted only if its CUSUM statistic clears a gate fixed on the development split — and takes the largest per-segment mean.

At the block level the entire choice is worth 0.007 AUROC, and the development-selected logit mean is not the optimum on the mixed cells, where plain mean, top-25% and $K{=}8$ are 0.004–0.013 higher; we keep the development choice rather than reporting the test-optimal one. At the token level

Table 13: Line-head read-out. Panel 1: aggregations of the per-line logits of the line head reported in Table 1, with the $K{=}16$ block head added as the other path over the same token profile. Panels 2 and 3: the learned attention aggregator, with and without the sinusoidal positional encoding. Each panel is compared against *its own* line-mean row, because the line head has to be retrained for each of the three runs and cuDNN GRU kernels are not bitwise reproducible: per-file scores of two retrainings correlate at 0.99 but are not identical, so absolute values are comparable within a panel and not across panels. Columns, protocol and * are as in Table 2.

	All cells		Mixed cells		TPR@1%FPR	
Read-out	AUROC	Δ	AUROC	Δ	%	Δ
Panel 1: line head of Table 1						
line mean (ref.)	0.836	–	0.751	–	37.4	–
line max	0.832	-0.004^* [−0.007,−0.001]	0.756	+0.005 [−0.000,+0.010]	37.4	+0.0 [−2.8,+3.5]
block head $K{=}16$	0.853	$+0.017^*$ [+0.011,+0.024]	0.749	−0.003 [−0.013,+0.007]	48.4	$+11.0^*$ [+6.3,+14.2]
Panel 2: retrained line head, positional encoding on						
line mean (ref.)	0.833	–	0.750	–	35.8	–
attention aggregator	0.823	-0.009^* [−0.013,−0.006]	0.752	+0.002 [−0.002,+0.006]	24.2	-11.6^* [−15.4,−7.1]
Panel 3: retrained line head, positional encoding off						
line mean (ref.)	0.832	–	0.749	–	37.9	–
attention aggregator	0.832	−0.001 [−0.003,+0.001]	0.757	$+0.007^*$ [+0.005,+0.010]	35.2	−2.7 [−18.7,+1.3]

the choice matters at the operating point rather than in AUROC: the change-point tail mean keeps 0.820 AUROC but loses more than half of the TPR at 1% FPR ($47.1 \rightarrow 22.4$), whereas the best-segment variant is indistinguishable from the plain mean. Neither of the two localization-flavoured aggregations of the token score buys anything over averaging it, which is the reason the block head, and not a segmentation of the token sequence, is our attribution read-out.

Line-head read-out and its positional information. The line head consumes the same token profile as the block head and differs only in supervision granularity, so its read-out is ablated separately (Table 13). The learned aggregator over line states is a small Transformer, which is equivariant to a permutation of the lines: the position of a line inside the file can only reach it through the sinusoidal encoding, and removing that encoding is therefore the positional ablation on this path.

Averaging the line logits is the best read-out of the line head: the max is 0.004 below it and the learned aggregator 0.009 below, the latter losing a third of the TPR at 1% FPR. Removing the positional encoding does not hurt the aggregator — it closes the gap to the line mean (-0.001, interval containing zero) and gains 0.007 on the mixed cells — so on this path too the model is not using where in the file a segment sits. Panel 1 also quantifies the difference between the two supervision granularities at equal input: the block head is 0.017 AUROC and 11.0 TPR points above the line head averaged over all cells, while on the mixed cells the two are indistinguishable (-0.003, interval containing zero), which is the same trade-off visible per cell in Table 1.

How the block supervision is constructed. Section 3.4 makes two choices that can be ablated by retraining the head on a different corpus: mixing two synthesis forms, and keeping the paired human original of every mixed file as a negative. Table 14 does that, and its own training diagnostics are the reason we read it as a data-volume result rather than as a construction result.

The diagnostics have to be read before the Δ column. The reference stops improving on development after 4 epochs at a block-level AUROC of 0.914; *every* reduced-corpus run peaks after a single epoch at 0.62–0.82 and then descends monotonically while its training loss falls to 0.003, and pinning all runs to 4 epochs reproduces the same development AUROC to three decimals. The head therefore memorizes a reduced corpus, which means these differences conflate the construction with an optimisation failure, and we report them as a sensitivity to corpus volume rather than as the effect of the construction.

Within that limit one comparison is still clean. *Continuation only* ($11{,}111$ pure blocks) is *above* the volume-matched subsample that keeps both forms ($10{,}807$ blocks): 0.730 against 0.683. We therefore have no evidence that mixing continuation with fill-in-the-middle supervision is what the block head needs; what the numbers support is the weaker statement that it needs volume ($17{,}767$ blocks $\rightarrow 0.853$, roughly $11{,}000$ blocks $\rightarrow 0.68$–0.73 regardless of form). Removing the paired

Table 14: Supervision-construction ablation for the $K{=}16$ block head. Every row retrains the head with the same features and hyper-parameters on a different training corpus, for a fixed 4 epochs (the reference's early-stopping point), so that the rows differ in data and not in schedule; the development split is unchanged and is used only to pick the aggregation. *Pure blocks* counts the blocks that are entirely AI or entirely human, i.e. those that enter the loss, and *dev* is the block-level AUROC on the development split. *Corpus* 63% is a hash-selected subsample chosen to match the block count of *continuation only*, which is the only pair in this table that separates the synthesis form from the corpus volume. Test columns, protocol and * follow Table 2.

	Training corpus			All cells		Mixed	TPR@1%
Corpus	files	pure blocks	dev	AUROC	Δ	AUROC	%
full corpus (ref.)	3,164	17,767	0.914	0.853	–	0.749	48.4
continuation only	1,686	11,111	0.818	0.730	-0.123^* $[-0.133,-0.114]$	0.620	8.8
fill-in-middle only	1,478	6,656	0.620	0.583	-0.271^* $[-0.284,-0.259]$	0.492	1.6
corpus halved	1,582	8,637	0.754	0.686	-0.168^* $[-0.178,-0.157]$	0.556	10.1
corpus 63%	1,994	10,807	0.768	0.683	-0.170^* $[-0.181,-0.161]$	0.558	8.0
no paired originals	1,582	7,220	0.716	0.777	-0.077^* $[-0.085,-0.070]$	0.669	24.8

Table 15: Alignment between block boundaries and true AI segments on the three mixed cells, for four partitions of the same files. The first three columns are shares of *blocks*: a block is pure-AI or pure-human if all of its lines have that label, mixed otherwise. *AI lines captured* is the median over files with at least one AI line of the fraction of that file's AI lines that lie inside all-AI blocks — the part of the signal a block-level head can see undiluted — and *diluted* is the share of AI lines that sit in mixed blocks, pooled over the whole cell and weighted by AI lines. The two columns are computed on different bases (a median over files against a line-weighted pool over the cell), so they are not complements of each other and do not sum to 100. A captured median of 0.0 means that in more than half of that cell's files no block is entirely AI. *AST* is the declaration partition, *AST+K8 gaps* re-cuts its gap blocks on the 8-line grid. Files that `tree-sitter` cannot parse (1.5–2.1% per cell) or that contain no declaration (31–36%) fall back to one block per file and are counted as they are. No head is trained here: these are properties of the partition.

Partition	pure-AI %	pure-human %	mixed %	AI lines captured %	diluted %
COMPL. ($n = 1{,}496$)					
$K = 8$	50.1	43.0	6.9	92.3	6.5
$K = 16$	48.5	37.2	14.3	82.4	13.4
AST	51.8	36.6	11.6	11.8	50.8
AST+K8 gaps	49.7	44.1	6.2	90.5	18.3
REFAC. ($n = 1{,}899$)					
$K = 8$	8.8	34.5	56.7	0.0	72.9
$K = 16$	5.3	23.8	70.9	0.0	83.8
AST	6.2	48.6	45.2	0.0	91.4
AST+K8 gaps	9.2	46.2	44.6	0.0	77.9
FIM ($n = 1{,}300$)					
$K = 8$	24.0	36.1	39.9	28.1	44.5
$K = 16$	13.7	27.2	59.1	0.0	66.9
AST	27.1	38.8	34.1	0.0	81.5
AST+K8 gaps	27.4	40.8	31.9	10.5	57.0

human originals costs 0.077 AUROC and half of the TPR, in the direction Section 3.4 predicts, but that corpus is also the smallest of the six, so that row does not separate the paired negative from the volume either. One further choice is not load-bearing: weighting the loss so that every file contributes equally, rather than every block, changes the average AUROC by $+0.001$ and the TPR by $+0.0$ points.

Syntax-aligned blocks: a negative result. Fixed K-line blocks cut across syntax, so the natural alternative is to make one block per declaration. We built that partition with `tree-sitter` — one block per terminal declaration, container declarations expanded into their members, and the lines between declarations merged into *gap* blocks — and measured alignment directly, without training a head, on the three mixed cells (Table 15). It does not align better than the fixed grid.

The pure declaration partition is unusable: on COMPL. it captures 11.8% of the AI lines in all-AI blocks against 92.3% for $K = 8$, because 55–59% of all lines fall between declarations and are swallowed by long gap blocks. Re-cutting the gaps on the 8-line grid fixes that (90.5%) and does lower the mixed-block share on the two cells where it is high (44.6 against 56.7 on REFAC., 31.9 against 39.9 on FIM), but it never captures more AI lines than the plain grid on any of the three cells. The reason is that the AI segments of this corpus are not declaration-shaped: of the 688 AI segments in the Python files of FIM, none coincides with a declaration, 345 lie strictly inside one, and 343 cross a declaration boundary or fall outside every declaration — the hole positions are drawn to hit a target AI ratio, and the refilled text shifts every line number after it. We therefore keep the fixed grid. This is a statement about this benchmark, not about the idea: a corpus of whole-function completions would align with declarations by construction and is not measurable here.

Backbone swap. The deepseek-coder-1.3B row of Table 2 uses the same recipe, training files and hyperparameters as the Qwen2.5-Coder-1.5B reference, and the Δ is significant under the paired bootstrap. Its tokenizer cuts the same files into 27% more tokens, so the row is restricted to the token-ratio file-mean read-out; its base NLL on this corpus is also visibly lower than Qwen's.

A.6 FORMATTING CONTROLS

Table 16 is the evidence behind Section 5.4, and it bounds the formatting explanation from two sides. The left panel is a static ceiling: a logistic probe trained on the same corpus as the supervised baselines, restricted to whitespace features, reaches only 0.64–0.67 AUROC on the pure cells, 0.54 on COMPL., and chance level on FIM; with file length alone the probe is near chance on every cell.

The right panel intervenes on the input instead: we erase formatting from the files and re-score them with the unchanged detector. Two levels of normalization are applied, in the following order.

Soft unifies line endings (`CRLF` and lone `CR` become `LF`), then strips trailing spaces and tabs from every line, then collapses each run of blank lines to a single blank line and drops the file's leading and trailing blank lines. Indentation and intra-line whitespace are left untouched, so the code remains syntactically valid in any language.

Hard applies *Soft* first and then, line by line, removes all leading spaces and tabs, collapses every remaining run of spaces and tabs inside the line to a single space, and deletes blank lines entirely. This destroys every whitespace statistic — indentation width, blank-line structure and alignment alike (Python files lose their block structure; the scorer reads the file as text and does not require compilable code).

Every operation at both levels only deletes or merges space, tab and newline characters and never touches a non-whitespace one, which is why both levels provably leave the non-whitespace character sequence identical to the original file. Erasing formatting entirely leaves the detector far above the whitespace ceiling of the left panel and above every zero-shot method, at the cost read off in Section 5.4; the one cell that moves the other way is FIM, which *improves* by 1.7 points, indicating that indentation was acting as noise there. The detector reads the code, not the layout.

A.7 FALSE POSITIVES ON REAL REPOSITORY CODE

Table 18 is the evidence behind Section 5.5, and Table 19 carries the block- and line-granularity rates quoted at the end of that section. Two threshold regimes are reported because they answer different questions: the 1% FPR operating point is the deployment setting used throughout the paper, and the accuracy-maximising threshold is the one behind Table 6 (Appendix A.2). The column "synthetic" repeats the same computation on the $2{,}541$ human files of the internal test set, so the two columns isolate the effect of changing the negative pool from synthetic to real repository code.

At the deployment operating point all four read-outs stay at or below 0.81%, against a nominal 1% on the development split and 0.35–0.87% on the synthetic negatives, and the block head at $K = 16$ is the most conservative read-out on both pools. The accuracy-maximising threshold sits in the middle of the score distribution by construction and therefore gives 4.7–10.0%; that regime is reported for completeness, not as an operating point. Per-language rates are read off the token ratio, the only read-out whose ordering is stable: JavaScript (0.86%), PHP (1.42%) and Ruby (1.23%) at the loose end and Go (0.16%), Java (0.28%) and Python (0.33%) at the tight end, the same ordering

Table 16: Is the signal formatting? *Left panel:* logistic probes trained on the same corpus as the supervised baselines, restricted to nine whitespace features or to length alone — the static ceiling. *Right panel:* the file-mean token-ratio detector of Table 1, adapters and aggregation unchanged, re-scoring the test files after the two levels of formatting normalization defined above; both leave the non-whitespace character sequence identical, so any change is attributable to the erased formatting. Δ is *Hard* minus *Original*, computed from the unrounded AUROCs, so it can differ by 0.001 from the difference of the two rounded entries. All entries are threshold-free file-level AUROC against the shared HUMAN cell, with no threshold chosen anywhere. The feature list is in Appendix A.9.

	Static probe (ceiling)		Detector, formatting erased			
Cell	Whitespace only	Length only	Original	Soft	Hard	Δ (Hard−Orig.)
BASE	0.650	0.553	0.964	0.958	0.925	-0.039
UNSEEN	0.643	0.561	0.968	0.962	0.941	-0.028
REWRITE	0.666	0.564	0.972	0.963	0.914	-0.058
COMPL.	0.539	0.603	0.711	0.699	0.675	-0.036
REFAC.	0.600	0.539	0.810	0.789	0.741	-0.069
FIM	0.504	0.550	0.601	0.600	0.619	$+0.017$
Avg.	0.600	0.562	0.838	0.828	0.802	-0.035

Table 17: False-positive rate (%) per language at the deployment operating point (the 99th-percentile development threshold of Table 18, transported unchanged), on the same $19{,}445$ pre-2022 repository files. Every language with at least 100 files is tabulated, in descending order of file count; languages below that cutoff are pooled into *other* (279 files). The block head's language feature covers 14 languages; Dart, Objective-C, Swift, Kotlin, Lua and Rust — and everything in *other* — fall outside it. Lower is better; a rate of 0.00 is an exact zero, not a missing entry.

Language	n	Token ratio	Block $K{=}8$	Block $K{=}16$	Line head
JavaScript	4,545	0.86	0.84	0.68	1.67
Java	3,173	0.28	0.44	0.28	0.35
Python	3,038	0.33	0.30	0.13	0.39
C	1,721	0.41	1.51	2.21	0.58
C#	1,190	0.76	1.60	0.17	0.67
PHP	1,125	1.42	0.44	0.00	0.71
C++	861	0.70	0.35	0.35	0.58
TypeScript	797	0.38	0.13	0.00	0.88
Ruby	652	1.23	0.46	0.15	0.61
Go	614	0.16	0.98	0.49	0.65
Shell	382	0.26	0.00	0.00	0.00
other	279	0.36	0.72	0.36	0.36
Dart	255	0.00	3.53	**4.31**	1.18
Objective-C	213	1.88	1.41	0.94	3.29
Swift	173	1.73	0.00	0.00	0.58
Kotlin	154	0.00	0.00	0.00	0.00
Lua	140	0.00	0.71	0.71	0.71
Rust	133	0.00	0.75	0.00	0.00

as on the synthetic pool. The unit-level heads do not preserve it — Go is the loosest of the five tabulated languages under the $K = 8$ head (0.98%) and the tightest but one under the token ratio — so the ordering is a property of the language *and* the read-out rather than of the language alone. The full per-language panel is Table 17: every language with at least 100 files, all four read-outs, at the deployment operating point. It carries the two facts the five-language slice cannot: the heads disagree about which languages are hard (C-family C, C# and Dart are where the $K{=}16$ head concentrates its worst rates, while the token ratio is loosest on PHP, Ruby and Objective-C), and every read-out has a language where its rate is at least $3\times$ its aggregate — 3.29% on Objective-C for the line head, 2.21% on C for the $K{=}16$ head against a 0.55% aggregate, 3.53% on Dart for the $K{=}8$ head. One caveat on coverage: the block head's language feature covers 14 languages and everything else falls into an *other* bucket, and the worst per-language rate at the deployment operating point — Dart at $K = 16$ (4.31%, $n = 255$) — sits in that bucket, so it is a read-out-specific failure rather than a language the corpus never showed the head.

Table 18: False-positive rate (%) on 19,445 pre-2022 repository files. All files are negatives, so only FPR is defined. Every threshold is calibrated on the development split (2,031 human files) and transported unchanged: the upper panel uses the 99th percentile of the development human scores (a nominal 1% FPR on development, transported here rather than read off the test HUMAN cell as in Table 1), the lower panel the accuracy-maximising threshold. "Synthetic" applies the same threshold to the 2,541 human files of the internal test set and is the reference point for the "all" column. Lower is better everywhere; per-language file counts are in Appendix A.9.

Read-out	threshold	synthetic	all	Python	Java	JavaScript	TypeScript	Go
1% FPR operating point (deployment)								
Token ratio, file mean	0.0116	0.67	0.60	0.33	0.28	0.86	0.38	0.16
+ block head, $K = 8$	0.383	0.63	0.72	0.30	0.44	0.84	0.13	0.98
+ block head, $K = 16$	−0.303	0.35	**0.55**	0.13	0.28	0.68	0.00	0.49
+ line head, file mean	1.364	0.87	0.81	0.39	0.35	1.67	0.88	0.65
Accuracy-maximising threshold (reported for completeness)								
Token ratio, file mean	0.00081	5.08	4.73	3.62	2.40	6.58	4.89	1.79
+ block head, $K = 8$	−1.434	7.71	8.66	3.46	7.03	11.11	6.15	9.93
+ block head, $K = 16$	−2.107	5.67	7.31	3.23	4.88	8.84	4.64	6.35
+ line head, file mean	−0.100	11.29	9.99	7.04	8.70	14.30	11.79	6.35

Table 19: False-alarm rate (%) at block and line granularity, on the same 19,445 pre-2022 repository files as Table 18. Every file is a negative, so every block and every line inside it is a negative too and only FPR is defined: n is the number of blocks (or lines) scored, not the number of files, and the rate is the share of those units above threshold. Thresholds are the granularity-matched thresholds frozen on the development split and transported unchanged — the block-level threshold of Table 7 (0.3214 at $K = 8$, 0.2271 at $K = 16$) and the line-level threshold of Table 8 (−0.272) — so they are *not* the file-level operating points of Table 18 and the two tables are not comparable row by row. The token ratio has no sub-file read-out and is absent. 53,525 lines are not covered by the line head and are excluded from its denominator. Lower is better.

Read-out	threshold	n	FPR (%)
Block head, $K = 8$ (blocks)	0.3214	259,682	9.90
Block head, $K = 16$ (blocks)	0.2271	134,703	8.89
Line head (lines)	−0.272	1,955,822	17.33

Table 20: Internal dataset (`aicd_0907`). Every split is a set of files labelled human or AI-involved; mixed files additionally carry character-level AI spans, from which both the line labels and the block labels are derived (Section 2). Pairs count as two files: train / part1 is 6,001 pairs, dev / pure 2,031 pairs, and train / part2 is 1,582 mixed files (843 continuation and 739 fill-in-the-middle) each accompanied by its own human original. The test set applies one generation axis at a time (*cells*) to a fixed human base: 11,773 positives over six generation cells against one shared pool of 2,541 human negatives.

Split	Content	#files	Used for
train / part1	pure human/AI pairs	12,002	the two LoRA adapters
train / part2	mixed files + their human originals	3,164	the block head
dev / mixed	mixed files	398	K, head hyper-parameters
dev / pure	pure human/AI pairs	4,062	aggregation and threshold
test	6 generation cells + shared HUMAN pool	14,314	reporting only

A.8 Implementation details

The scoring backbone is Qwen/Qwen2.5-Coder-1.5B, kept frozen throughout. Each adapter is a rank-16 LoRA module with $\alpha = 32$ and 18.5M trainable parameters, targeting `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj` and `down_proj`, and trained with AdamW at learning rate 10^{-4} for one epoch on its own side of the pure-file split with a plain causal language modeling objective, truncating each file to 2048 tokens (the 90th percentile of the length distribution; 5–7% of training files are cut). Training and scoring run in fp16: the available accelerators are Volta-class and have no bf16 hardware, so bf16 would fall back to an emulation path. Batching is

Table 21: Head inputs. *Block head*: 88 columns per block of K lines, family prefixes as used by the drop-family rows of Table 2. *Line head*: 16 columns per token. $s.$ are the likelihood ratios of Section 3.1, H entropy, and dctx the long- minus short-context NLL gap, negated so that larger is more AI-like.

Family	Columns	#
Block head		
raw	$s_{\text{ai-hum}}$: mean, sd, min, max, $q_{10,25,50,75,90}$, positive rate	10
	$s_{\text{ai-base}}$, $s_{\text{base-hum}}$, nll_{base}: mean, sd, q_{50}	9
	H_{base}: mean, q_{50}; log-rank and top-1 rate: mean	4
	$-$dctx for the base, AI and human passes: mean	3
raw.size	log token count, line coverage, tokens and characters per line, block lines	5
z	within-file z-score of the 26 statistics above	26
ctx	offsets ±1, ±2: neighbour $s_{\text{ai-hum}}$ mean, difference, miss flag	12
pos	relative index, first/last flags, log block count	4
lang	one-hot over 14 languages and other	15
	total	88
Line head (per token; no pos, lang or size column)		
raw	nll of the base, AI and human passes; H_{base}; log-rank; the three short-context nll	8
z	within-file z-score of the same eight	8
	total	16

by token budget over length-sorted buckets, with gradient checkpointing enabled and no sequence packing.

Scoring is windowed rather than truncated. The per-token profile pass uses a 4096-token window with 512 tokens of carried context, which keeps files of any length scoreable. Training truncates at 2048 tokens while scoring reads 4096-token windows, so the positions and relative distances beyond 2048 are extrapolated at scoring time: the backbone's rotary embeddings are defined up to 32,768 positions, so this is a distribution shift rather than an out-of-range error, and every scored token still carries at least 512 tokens of real left context. The logit computation is sliced so that the vocabulary-sized intermediate never materializes for the whole window at once. The base-model pass additionally emits entropy and rank, and a second pass over a short context window supplies the short-context contrast features. Token scores are persisted at float32, so changing K does not require re-scoring.

The block head is a bidirectional GRU over the 88-dimensional block feature sequence, trained on GPU; mixed blocks are masked out of the loss. Its aggregator and K are chosen on the development split. Both heads use 2 layers of hidden size 64, dropout 0.1, Adam (Kingma & Ba, 2015) at 10^{-3}, at most 60 epochs with early stopping on the development split after 8 epochs without improvement, and seed 20260907; the block head batches 32 files at a time while the line head batches by a padded-token budget of 131,072, since single files of several thousand lines make a fixed file count overflow memory.

Head input features. Table 21 lists both heads' inputs. The block head's 88 columns form five families, named by the prefix that the drop-family rows of Table 2 remove. log-rank is $\log(1+\text{rank})$: the raw rank is long-tailed to 65,535, and on a linear scale a few large values dominate the block mean. Every signal statistic is given twice, once as a raw value and once as a within-file z-score whose mean and standard deviation are computed after 1%/99% winsorization, and the two are not redundant. With raw values only, how predictable a file is overall becomes a confound that masks between-block differences; with z-scores only, every block of a pure-AI file sits near its own file mean and is hard to separate from a block of a pure human file, while the file score is an aggregate of block logits. Removing either family is detectable but small on the test cells (-0.008 and -0.011 AUROC, Table 2), so this is a design argument, not a large effect. Blocks containing no token at all are dropped rather than zero-filled, which would enter the head as neutral evidence; for an out-of-range or empty neighbour in the ctx family the neighbour value falls back to the block itself, the difference is set to zero, and a miss flag is raised. The line head takes the same token profile but only 16 columns, and in particular no positional, language or size feature.

Feature definitions. Let T_b be the token set of block b, B the number of blocks and $\mathcal{B}_{\text{val}}$ the blocks holding at least one token. With $\text{nll}_y(t) = -\log p_y(x_t \mid x_{<t})$ for $y \in \{\theta, \text{ai}, \text{hum}\}$ and $\text{nll}^{\text{s}}_y(t)$ its short-context counterpart, the token signals behind both heads are

$$\begin{aligned} s_{\text{ai-hum}\,t} &= \text{nll}_{\text{hum}}(t) - \text{nll}_{\text{ai}}(t), \quad s_{\text{ai-base}\,t} = \text{nll}_{\theta}(t) - \text{nll}_{\text{ai}}(t), \\ s_{\text{base-hum}\,t} &= \text{nll}_{\text{hum}}(t) - \text{nll}_{\theta}(t), \qquad \text{dctx}^{y}_t = \text{nll}_{y}(t) - \text{nll}^{\text{s}}_{y}(t), \end{aligned} \tag{3}$$

for $y \in \{\theta, \text{ai}, \text{hum}\}$, all oriented so that larger means more AI-like: dctx is the context gain negated, since machine-written tokens gain less from surrounding context. The `raw` family reduces a signal v over T_b with

$$\begin{aligned} &\bar{v}_b = \frac{1}{|T_b|}\sum_{t\in T_b} v_t, \qquad \text{sd}(v_b) = \Big(\frac{1}{|T_b|}\sum_{t\in T_b}(v_t - \bar{v}_b)^2\Big)^{1/2}, \\ &\min_{t\in T_b} v_t, \quad \max_{t\in T_b} v_t, \quad q_p(v_b), \end{aligned} \tag{4}$$

q_p being the empirical p-quantile over T_b, $\text{pos}(v_b)$ the share of its tokens with $v_t > 0$, and r_t the rank of x_t under p_θ; Table 21 names which of these each signal takes. The `z` family standardizes each of the 26 raw statistics across the file,

$$z(v_b) = \frac{\text{clip}(v_b,\, w_{0.01},\, w_{0.99}) - \mu_v}{\max(\sigma_v, \epsilon)}, \qquad \epsilon = 10^{-6}, \tag{5}$$

where $w_{0.01}$ and $w_{0.99}$ are the 1% and 99% quantiles of $\{v_{b'}\}_{b'\in\mathcal{B}_{\text{val}}}$ and μ_v, σ_v the mean and standard deviation of those winsorized values, the numerator being winsorized the same way; a constant column yields 0 rather than an infinity. The `raw.size` family adds

$$\log(1{+}|T_b|), \quad \text{cov}_b = \frac{|\{\ell \in b : \ell \text{ carries a token}\}|}{K}, \quad \frac{|T_b|}{K}, \quad \frac{1}{K}\sum_{\ell\in b} c_\ell, \quad K, \tag{6}$$

with c_ℓ the character length of line ℓ including its newline. For each $\delta \in \{-2, -1, +1, +2\}$ the `ctx` family carries the neighbour mean $\bar{s}_{b+\delta}$, the contrast $\bar{s}_b - \bar{s}_{b+\delta}$ and a miss flag; when $b + \delta$ leaves $\{1, \ldots, B\}$ or is empty, the neighbour value falls back to $\bar{s}_b$, the contrast is 0 and the flag is 1. The `pos` family is $\big((b-1)/\max(B-1, 1),\ \mathbb{1}[b{=}1],\ \mathbb{1}[b{=}B],\ \log(1{+}B)\big)$, and `lang` is a one-hot over the 14 corpus languages plus *other*, identical for every block of a file.

The line head reads the same profile at token granularity. Its eight signals $[\text{nll}_\theta, \text{nll}_{\text{ai}}, \text{nll}_{\text{hum}}, H_\theta, \log(1{+}r), \text{nll}^{\text{s}}_\theta, \text{nll}^{\text{s}}_{\text{ai}}, \text{nll}^{\text{s}}_{\text{hum}}]$ are each fed twice, as the raw value and as the within-file z-score of Equation 5 taken over all tokens of the file rather than over blocks, which gives the 16 columns of Table 21. It adds no positional, language or size column, and its only within-line aggregation is the mean of the token logits over the tokens assigned to that line.

The two LoRA adapters use learning rate 10^{-4} and truncation length 2048 tokens; the block and line heads use Adam at learning rate 10^{-3}. The classification-head control is trained and scored with the same 4096-token window and 512-token carried context as the windowed scorer (stride $4096 - 512 = 3584$; its file score is the mean over window logits), so its context is no longer shorter than ours, as stated in Section 5.

A.9 Table conventions and protocol details

Unless a caption says otherwise: each cell reads AUROC / TPR@1%FPR / macro F1, the latter two in percent; on the internal test set the negative pool is the shared HUMAN cell (2,541 files) and the positive class is AI-involved with mixed files included ($\rho \geq 0.05$); *Avg.* or *all cells* is the unweighted mean over the six generation cells and *mixed* the mean over COMPL., REFAC. and FIM; chance is AUROC 0.500 and TPR 1%, while the macro F1 floor at the 1% FPR operating point is the all-negative predictor, which a score without signal approaches there; it depends on class balance and is quoted per caption, and the all-positive predictor is not used as a floor because its FPR is 100%; bold marks the best value in each column for each metric, and a dash means "not defined", not "not run". The threshold behind both operating-point metrics — TPR@1%FPR and, where it is reported at that point, macro F1 — is the 99th percentile of that table's own negative pool, which on the internal test set is the HUMAN cell of the test set and not a development quantity, and it is recomputed inside each replicate wherever a bootstrap interval is given. The ablation tables of Section 5.3 and Appendix A.5 inherit that convention; the tables whose thresholds are instead frozen on the development split (Tables 7, 8, 6, 17 and 18) say so in their captions.

Table 22: File-level detection on the internal test set, all three metrics per cell. This is Table 1 of the main text with AUROC and TPR@1%FPR restored; the macro F1 row of every cell and the *Avg.* column are identical to it. Metrics, negative pool, chance levels and bolding follow Section 4.4. The two operating-point metrics share the threshold read off the HUMAN cell, and the all-negative predictor scores 33.3–39.8% macro F1 across the cells. Group C rows are cumulative, not independent detectors.

	Method	BASE	UNSEEN	REWRITE	COMPL.	REFAC.	FIM	Avg.
		each cell, top→bottom: AUROC / TPR@1%FPR / macro F1 (%)						
		0.415	0.416	0.441	0.584	0.507	0.566	0.488
		0.2	0.1	0.1	0.7	0.3	1.3	0.4
	log-rank	33.3	33.2	35.7	39.1	36.5	40.9	36.5
		0.401	0.415	0.430	0.569	0.491	0.577	0.481
		0.2	0.1	0.1	0.5	0.2	0.7	0.3
A	LRR	33.4	33.2	35.7	38.9	36.4	40.3	36.3
		0.376	0.397	0.365	0.685	0.500	0.566	0.481
		0.2	0.2	0.2	3.3	0.4	0.5	0.8
	Fast-DetectGPT	33.3	33.3	35.9	41.8	36.6	40.1	36.8
		0.434	0.433	0.431	0.595	0.502	0.532	0.488
		0.1	0.1	0.1	0.7	0.5	0.8	0.4
	Binoculars	33.2	33.2	35.7	39.1	36.7	40.4	36.4
		0.466	0.477	0.486	0.583	0.530	0.569	0.518
		0.2	0.0	0.0	0.7	0.3	1.0	0.4
	DetectCodeGPT	33.3	33.1	35.7	39.1	36.5	40.6	36.4
		0.460	0.463	0.478	0.483	0.471	0.493	0.475
		1.0	0.8	0.8	1.2	0.2	1.6	0.9
	AIGCode Detector	34.1	33.8	33.8	34.3	33.2	34.7	34.0
		0.911	0.906	0.888	0.619	0.748	0.551	0.771
		35.5	32.9	25.1	2.0	7.1	1.5	17.3
	CodeT5+ + MLP	63.6	61.7	58.2	40.5	43.4	41.1	51.4
		0.943	0.909	0.939	0.618	0.770	0.566	0.791
		48.9	37.5	40.9	2.1	15.0	1.9	24.4
B	CodeBERT	72.2	64.9	69.1	40.6	50.6	41.6	56.5
		0.963	0.945	0.962	0.647	0.821	0.565	0.817
		53.1	43.0	46.4	3.3	16.1	2.0	27.3
	UniXcoder	74.7	68.5	72.5	41.8	51.5	41.7	58.4
		0.861	0.879	0.848	0.524	0.688	0.537	0.723
		26.3	27.4	18.6	2.3	4.4	1.6	13.4
	ImBD	57.0	57.8	53.1	40.8	40.8	41.3	48.5
		0.957	0.944	0.952	0.635	**0.854**	0.505	0.808
	Same backbone	61.1	48.0	50.8	3.7	17.8	1.7	30.5
	+ LoRA cls. head	79.3	71.6	75.0	42.2	52.9	41.3	60.4
		0.965	0.947	0.965	0.580	0.804	0.548	0.801
		67.6	64.0	57.2	5.2	20.2	2.5	36.1
	DroidDetect	82.9	80.9	78.6	43.7	54.9	42.2	63.8
		0.964	0.969	**0.972**	0.711	0.810	0.601	0.838
	Ours: token ratio,	**79.4**	80.1	**74.2**	12.0	31.3	5.7	47.1
	file mean	**89.1**	89.4	**87.4**	50.0	63.1	45.3	70.7
		0.940	0.967	0.924	0.805	0.806	**0.672**	0.852
C		74.0	82.6	65.9	**25.5**	30.9	**11.2**	**48.4**
	+ block head, $K = 8$	86.3	90.7	83.3	**60.8**	62.9	**50.4**	**72.4**
		0.954	**0.974**	0.944	0.789	0.819	0.638	**0.853**
		75.5	**84.2**	66.7	22.5	**32.3**	9.2	**48.4**
	+ block head, $K = 16$	87.1	**91.6**	83.7	58.5	**63.8**	48.6	72.2
		0.923	0.949	0.889	**0.817**	0.769	0.667	0.836
		62.0	67.7	46.3	20.6	20.1	7.6	37.4
	+ line head, file mean	79.8	82.9	72.4	57.1	54.8	47.1	65.7

Table 1 (internal test set). Table 22 below is the same comparison with all three metrics per cell; Table 1 in the main text carries its macro F1 rows and its *Avg.* column unchanged. Per-cell positive counts are BASE 2,541, UNSEEN 2,541, REWRITE 1,996, COMPL. 1,496, REFAC. 1,899 and FIM 1,300. The UNSEEN cell splits over its four writer models as 634 (Qwen3-Coder-Plus), 634 (Doubao-Seed-2.1-Pro), 637 (Kimi-K2.6) and 636 (MiniMax-M2.7). The aggregation is fixed on the development split and applied unchanged to test, and the 1% FPR point is read off the HUMAN cell itself; macro F1 uses that same operating point. The TPR@1%FPR average is tied at 48.4 between $K = 8$ and $K = 16$, so both are bold. The line head is a bidirectional GRU over the token signals emitting one score per line; its file score is the mean of its line logits, which beats both a max and a learned attention aggregator on the held-out development files. The AIGCODE row is scored on a stratified subsample of 500 files per cell (3,500 in total) rather than the full test set, and its composite score is our reconstruction: the original paper publishes neither the mask-weight function, nor a burstiness formula, nor the three weights. The DROIDDETECT row is a file-level ModernBERT classifier truncated to 2,048 tokens, not one of the chunk-level encoders it sits among.

Table 9 (segment length). Segments and block labels follow Section 2: a segment is a maximal run of AI lines after merging runs separated by at most one human line, and a block is positive if it contains at least one AI line. Every variant scores the same positive files against the shared HUMAN cell (2,541 files), except *oracle-pair*, which scores only the tokens inside the true AI spans and substitutes the same line range of each positive's paired human original for the negative, so that content difficulty is controlled. The *length-only* row is a logistic probe on line count and character count, trained on train split part 1. Positive and negative blocks are both drawn from the same mixed files. The file-level reading aggregates the same block logits by their mean against the shared human pool.

Table 16 (formatting). The whitespace probe uses nine features: indentation mean, std, max and number of distinct levels; tab-indented line ratio; blank-line ratio; longest blank run; trailing-whitespace line ratio; and line count. The length probe uses line count and character count. Both are trained on train split part 1. The probes are linear; tree ensembles can extract somewhat more from the same features.

Tables 18 and 17 (real human code). The five per-language columns of Table 18 are the five largest language slices of the corpus; Table 17 extends them to every language with at least 100 files, and its n column is the per-language file count. The two tables come from the same run and the same thresholds.

Table 3 (external transfer). Chunk scores (encoders) and block probabilities (our block heads) are aggregated to a file score by the aggregator selected on the corresponding development split: logit-mean for both K of the block heads, and mean for P1 against max for P2 on DroidCollection. The line head's file score is the mean of its per-line logits, which beats both a max and a learned attention aggregator on held-out development files. Chance for TPR@1%FPR is 0.010. Thresholds follow what each benchmark ships. TPR@1%FPR always reads the 99th percentile of the benchmark's own negatives. On AICD-Bench T1 and T3, which ship no development split, macro F1 is read at that same threshold, so those four F1 columns are calibrated on the test negatives and are optimistic in the sense discussed in the Reproducibility Statement; on DroidCollection, which does ship a development split, the macro F1 threshold is the value that maximizes pooled macro F1 there and is transported to test unchanged. The AIGCODE row is zero-shot and therefore has no P2; DROIDDETECT is a file-level classifier truncated to 2,048 tokens. ImBD scores the conditional-probability curvature of an SPO-aligned model over the full file, truncated to 8,192 tokens; P1 transports the internally aligned model unchanged. Its P2 needs one caveat. SPO is a DPO stage over *content-equivalent* pairs — the same file written by a human and rewritten by a model — and no external benchmark provides them: on all three, the machine-written and human sides are disjoint file sets with no shared identifier. We therefore pair within each benchmark by bucketing on (language, source) and matching on token length inside each bucket, which aligns the carrier distribution but not the content, and train on a quarter of the pairs (3,997 / 5,077 / 4,731 for T1 / T3 / DroidCollection) so that the SPO budget stays comparable to the 5,873 pairs used internally. Held-out DPO accuracy reaches 1.000 / 0.948 / 0.943, so the alignment fits its own objective; what it aligns to is partly corpus identity rather than style. Two consequences are visible in the table. On T3 adaptation *inverts* the ranking (0.399, italicized), the only sub-chance P2 entry anywhere; and on all three the direction check on held-out pairs comes out negative, i.e. the aligned model assigns *lower* curvature to machine code, so the sign is flipped before scoring. Read the ImBD P2 column as the cost of transplanting a protocol whose data assumption the target corpus does not satisfy, not as a ceiling on the method.

Table 4 (HybridCodeAuthorship). The line level carries 229,158 attribution runs for our methods and 215,254 for Group A, which drops runs the scoring tokenizer reduces to zero tokens; Group B scores 215,422 runs, and 215,258 for ImBD, which skips runs its tokenizer empties. The Block columns carry 350,755 (K=8) and 178,050 (K=16) blocks, scored for every row; each method is scored on the unit of the column it fills: a Line cell scores one attribution run and takes that line's own label, a Block cell scores the K-line block as a single text and counts it positive if the block contains at least one AI line, and a File cell scores the whole file, so AIGCODE (perturbation sampling) and DROIDDETECT fill all three. DetectCodeGPT was not run. HCA ships no training or development split, so no threshold in this table is calibrated on HCA itself. Our rows take their

Table 23: External benchmarks used in Section 5.6. *Granularity* is the unit at which the benchmark carries ground truth (we always report a file-level decision, and additionally match the native granularity where one exists). *#eval* is the number of files we score after subsampling, with the negative count in parentheses. Without a train split only P1 is defined. *Published* is the best number reported by the benchmark's own paper together with its metric, which is in general neither our metric nor our protocol; it is included only as a same-protocol reference for our P2 column, and n/r means the benchmark reports no comparable number — as opposed to the "–" that Section 4.4 reserves for "not defined".

Benchmark	Granularity	Native label	#eval (neg.)	Train split	Published (metric)
AICD-Bench T1	file	binary	20,000 (15,519)	yes	62.31 (macro-F1, zs-LLM)
AICD-Bench T3	file	4-class	20,000 (13,447)	yes	61.65 (macro-F1, FT)
DroidCollection	file	4-class	105,826 (49,932)	yes	97.00 (domain-avg wF1, 2-cls, FT)
HCA	line, chunk	per-line binary	10,488 (1,904)	no	56.0 (F1, line, zs)

The four DroidCollection classes are *human*, *machine-generated* (26,524), *machine-refined* (13,230) and *adversarial* (16,140); we map the latter three to $y = 1$ and evaluate them both pooled and separately. The published entries read as follows. AICD-Bench T1's 62.31 is a zero-shot chain-of-thought LLM (Gemini-2.5-Flash) on a 5% stratified subsample; the best detector *trained on the benchmark's own split* is DeBERTa at 34.13, below the random baseline of 45.73. T3's 61.65 (ModernBERT) and DroidCollection's 97.00 (DroidDetect-Large, binary, domain-averaged) were both obtained by training on the benchmark's own training split — the same protocol as our P2, but in macro/weighted F1 rather than AUROC.

Table 24: Full Group A (zero-shot) transfer results, summarized by the DetectCodeGPT row of Table 3. Group A trains nothing, so only the transfer protocol P1 is defined. Each cell is AUROC / TPR@1%FPR / macro F1 (%).

Method	AICD-Bench T1	AICD-Bench T3	DroidCollection
LogRank	0.665 / 5.5 / 48.8	0.626 / 0.0 / 40.0	0.626 / 3.3 / 35.4
LRR	0.663 / 2.9 / 46.3	0.656 / 1.3 / 41.4	0.628 / 2.0 / 34.0
Fast-DetectGPT	0.637 / 10.4 / 53.2	0.653 / 6.1 / 46.1	0.604 / 6.5 / 38.6
Binoculars	0.529 / 1.0 / 44.5	0.406 / 0.0 / 40.0	0.497 / 0.4 / 32.3
DetectCodeGPT	0.703 / 6.5 / 49.7	0.666 / 0.2 / 40.2	0.669 / 3.0 / 35.1
AIGCode Detector	0.505 / 1.8 / 45.2	0.438 / 1.1 / 40.7	0.510 / 2.7 / 34.8

line- and block-level thresholds from the internal development split, frozen there and transported unchanged — the line threshold is the -0.272 of Table 8, the block thresholds come from the internal $K{=}16$ development blocks — which is a granularity mismatch we record rather than repair, since those values were selected on internal lines and blocks and not on HCA attribution runs. The File column instead reads the 99th percentile of HCA's own negatives, so its $F1$ is calibrated on the test negatives in the same way as the internal macro F1 columns. Group B baselines are internal fine-tunes transported unchanged: encoders score tree-sitter chunks (the internal ≤ 512-token splitting) aggregated by mean, the LoRA head scores the 4096-token windows aggregated by mean, and ImBD scores the conditional-probability curvature of its SPO-aligned model over the full file; at the line level every Group B row scores each attribution run directly with its own file-level input recipe. At the file level the zero-shot direction inverts (AUROC far below 0.500): the purely human files average as *more* machine-like than the co-written ones, i.e. the detectors rank boilerplate-heavy human code as generated. At the file level the Group B rows sit at chance-to-weak (0.472–0.594), apart from DroidDetect at 0.712; at the line level CodeBERT is the best of them at 0.666, below our block head at $K = 8$ (0.685), while the LoRA head *inverts* to 0.337 and ImBD reaches only 0.536. The two are not at the same granularity, so the comparison places our block read-out at the upper end of the baselines rather than clear of them.

A.10 External benchmarks: composition and per-slice results

Table 23 fixes what each external benchmark contributes, and Table 4 reports HybridCodeAuthorship at all four granularities.

Two entries in Group C of Table 4 need their own reading. First, the line column: the line head is the strongest of our read-outs there (0.589 / 10.6 / 62.5 against 0.528 / 0.9 / 53.5 for the token ratio, the only other read-out defined at that granularity), and its $F1$ is an order of magnitude above

Table 25: Full version of Table 4: HybridCodeAuthorship (Patterson et al., 2026) under P1 with every row restored, i.e. the zero-shot statistics and the CodeT5+ + MLP baseline that the main-text table omits. Metrics, negative pool, chance levels and bolding follow Section 4.4. F1 here is plain, not macro. Every entry shared with Table 4 is identical to it.

	Method	HCA, P1 (zero adaptation)			
		Line	Block K=8	Block K=16	File
		AUROC / TPR@1%FPR / F1 (%)			
A	LogRank	0.509 / 0.6 / 1.1	0.592 / 0.3 / 0.6	0.587 / 0.3 / 0.6	0.007 / 0.0 / 0.0
	LRR	0.491 / 0.3 / 0.6	0.583 / 1.2 / 2.3	0.581 / 1.3 / 2.5	0.050 / 1.1 / 2.2
	Fast-DetectGPT	0.495 / 1.2 / 2.2	0.568 / 1.8 / 3.5	0.568 / 1.9 / 3.7	0.499 / 0.0 / 0.0
	Binoculars	0.474 / 0.3 / 0.6	0.538 / 0.2 / 0.4	0.523 / 0.2 / 0.3	0.149 / 0.0 / 0.0
	Binoculars (reverse)	0.550 / 0.9 / 1.8	0.551 / 0.4 / 0.8	0.537 / 0.2 / 0.5	0.124 / 0.0 / 0.0
	AIGCode Detector	0.646 / 1.4 / 2.7	0.513 / 2.2 / 4.1	0.501 / 1.7 / 3.2	0.496 / 1.4 / 16.7
B	CodeT5+ + MLP	0.572 / 2.8 / 5.5	0.566 / 2.3 / 4.3	0.586 / 2.6 / 5.0	0.570 / 2.1 / 4.1
	CodeBERT	**0.666** / 6.1 / 11.4	0.577 / 4.8 / 9.0	0.589 / 5.4 / 10.1	0.594 / 3.3 / 6.3
	UniXcoder	0.581 / 3.6 / 6.8	0.562 / 4.8 / 9.0	0.570 / 5.2 / 9.8	0.580 / 2.4 / 4.6
	Same backbone + LoRA cls. head	*0.337* / 0.6 / 1.1	0.532 / 1.3 / 2.5	0.534 / 1.4 / 2.7	0.476 / 1.2 / 2.4
	ImBD	0.536 / 1.3 / 2.5	0.531 / 2.6 / 5.0	0.553 / 3.0 / 5.7	0.472 / 2.2 / 4.3
	DroidDetect	0.586 / 2.6 / 5.0	0.682 / 6.6 / 12.2	**0.685** / 9.3 / 16.8	0.712 / 11.3 / 26.7
C	Ours: token ratio, line mean	0.528 / 0.9 / 53.5	0.637 / 6.9 / 49.5	0.628 / 9.2 / **52.1**	0.638 / 12.8 / 22.7
	+ block head, $K = 8$	–	**0.685 / 14.0 / 51.5**	–	0.690 / 15.3 / 26.5
	+ block head, $K = 16$	–	–	0.627 / **11.0** / 50.4	0.667 / 14.4 / 25.2
	+ line head, run mean	0.589 / **10.6** / **62.5**	–	–	**0.786 / 19.1 / 32.0**

every baseline's. Second, the file column: the line head is the strongest row there (0.786 against 0.690 / 0.667 for the block heads, 0.638 for the token ratio and 0.712 for DroidDetect, the strongest baseline), so the usable margin on HCA comes from aggregating per-line evidence over the file rather than from reading it line by line. The block columns are read the same way as the line column: at a median span of two lines almost every K-line block contains both authors, so a block label that fires on a single AI line is close to a file-level label smeared over blocks, and the block AUROC drops from 0.685 at $K = 8$ to 0.627 at $K = 16$ as the blocks get coarser.