Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Position, not importance: what a likelihood threshold actually defers on

SHUBHANANDAN T Y

shubh442005@gmail.com

DSU: Dayananda Sagar University https://orcid.org/0009-0006-2182-4510

Mohan V S

Dayananda Sagar University https://orcid.org/0009-0003-4421-5281

Bharath Kumar Shanmugam

Dayananda Sagar University https://orcid.org/0000-0003-4535-3697

Puneeth N

Dayananda Sagar University https://orcid.org/0000-0002-2683-5087

Research Article

Keywords: Code generation, Learning to defer, Assistive agents, Empowerment, Surprisal, Human-AI interaction

Posted Date: October 5th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11241002/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Position, not importance: what a likelihood threshold actually defers on

Shubhanandan T Y [1*], Mohan V S [1],
Bharath Kumar Shanmugam [1], Puneeth N [1]

[1*]Department of AI & Robotics, Dayananda Sagar University, Bengaluru, Karnataka, India.

*Corresponding author(s). E-mail(s): shubh442005@gmail.com;
Contributing authors: vsmohan712@gmail.com;
shanmugabharathkumar@gmail.com; puneethn0015@gmail.com;

Abstract

Empower, a recent assistant-training method, teaches a language model to stop writing where its continuation stops being predictable, assuming predictable means unimportant. We test this on code. CEDE-Bench has 124 hand-authored items: 112 points where a Python completion chooses a dangerous or safe branch, and 12 benign controls with no consequential choice. At $\eta = 4$, the user-study threshold, its stopping rule writes the dangerous branch on 37 of 112 decisions (33.0%, Wilson 95% confidence interval 25.0 to 42.2) and on 8 of 10 worst-case probes. On one item it completes eighteen of nineteen tokens of an f-string SQL query. At the same threshold it cuts into 11 of 12 benign controls, finishing about a third of boilerplate. Its stopping point is statistically indistinguishable between consequential and benign items (mean completed fraction 0.26 against 0.35, Mann–Whitney $p = 0.16$). Position decides it. The commit rate correlates with the decision token's index at $r = -0.56$ and with its probability at $r = +0.26$; it is 76% for decisions in the first two tokens and 0 of 35 past token ten. Under a roughly constant entropy rate, the boundary reduces to $\eta \ln 2/H$, a token count set by local predictability; this recovers the observed stop with $R^2 = 0.55$ and a median error of one token. At $\eta = 0.32$ the commit rate falls to 8.9%, but only 7.5% of benign text gets completed, so no point on the sweep is both safe and useful. Data and code are released.

Keywords: Code generation, Learning to defer, Assistive agents, Empowerment, Surprisal, Human-AI interaction

1 Introduction

A code assistant has to keep deciding, without being asked, how much to write before it gives the keyboard back. If it writes too much, it makes choices that belonged to the programmer. If it stops too soon, it is not much help.

Empower (Ellis et al, 2025) answers this question without a reward model, preference labels, or human annotation. The recipe is short. A base model scores every token of a continuation, and the method keeps the longest prefix whose cumulative likelihood stays above a threshold. The assistant is then trained on those truncated prefixes. Behind the recipe sits a claim: a continuation becomes hard to predict exactly where a real decision is being made, so a likelihood threshold should find decision boundaries for free.

The reported results are strong. Simulated programmers helped by the method reach a 192% higher success rate than a supervised baseline. In an 18-person user study, participants preferred the empowerment assistant 78% of the time ($p = 0.015$), accepted 31% more of its completions, and received 38% fewer suggestions overall.

We think the premise is wrong in a way this evaluation could not see, and that code is the domain where it is most wrong. In real code the dangerous branch is usually the ordinary one. People paste `verify=False` when TLS gets in the way. The tutorial says `allow_origins=["*"]`. A model completing a function named `hash_password` expects `hashlib.md5`, because that is what its training corpus contains.

There is direct evidence for this. Pearce et al (2022) found that roughly 40% of Copilot completions in security-relevant scenarios were vulnerable. Perry et al (2023) found that participants who had an AI assistant wrote less secure code while feeling more confident that it was secure. Later benchmarks reproduced the pattern at larger scale, across models and languages (Siddiq and Santos, 2022; Hajipour et al, 2024; Bhatt et al, 2023). If insecure idioms are the high-likelihood continuation, then a rule that completes high-likelihood tokens will complete them without pausing. That is the one behaviour the method exists to prevent.

This paper measures how often it happens. We built CEDE-Bench, a set of 112 consequential coding decisions across five risk categories, together with 12 matched benign controls. We ran the Empower boundary rule over it at both published operating points and across the range between them. The study is organised around four research questions.

RQ1. How often does the boundary rule commit the dangerous branch of a consequential coding decision? (Sections 4.1 and 4.2)
RQ2. Does the same threshold also cut into benign code that contains no decision? (Section 4.3)
RQ3. Which properties of an item predict whether its decision is committed, and is importance one of them? (Sections 4.4, 4.5 and 4.7)
RQ4. Can the point where the rule stops be explained analytically? (Section 4.6)

The question matters for software engineering practice. Where a completion stops decides which choices reach the programmer as choices and which arrive already made, and Section 2 reviews evidence that programmers under-check the completions they accept (Barke et al, 2023; Parasuraman and Riley, 1997). In brief, our answers to the four questions are as follows.

First (RQ1), at $\eta = 4$ the rule commits the dangerous branch on 28.4% of standard items and on 80% of probe items. Second (RQ2), at that same threshold it stops partway through 11 of 12 benign controls, so the two failure modes are not something one dial can trade

against each other. Third (RQ3), whether a decision gets committed is not predicted by how dangerous it is, and only secondarily by how predictable it is. The main predictor is how early in the continuation the decision sits. Fourth (RQ4), this dependence on position has a closed form. Per-token surprisal in text is approximately constant (Genzel and Charniak, 2002; Jaeger, 2010), so a threshold on cumulative likelihood acts, to a good approximation, as a token counter whose count is set by the local entropy rate. We derive that count, test it, and recover 55% of the variance in where the rule actually stops.

The fourth result is the one we care about most. It says what the rule is, rather than only where it fails. Importance never enters the mechanism, and no amount of threshold tuning will put it there.

2 Related work

Empowerment as an assistance objective

Empowerment started as an information-theoretic measure of how much control an agent has over its own future observations (Klyubin et al, 2005; Salge et al, 2014). Variational lower bounds later made it practical to learn (Mohamed and Rezende, 2015). Du et al (2020) turned it into an objective for assistance: instead of inferring the human's reward and optimising it, help the human by enlarging the set of futures they can reach. Myers et al (2024) learn assistive behaviour from contrastive successor features, with no reward inference at all. Empower (Ellis et al, 2025) carries the idea into language models by swapping the empowerment estimate for a likelihood threshold over token sequences.

Before we disagree with part of Empower's case, we want to state it at full strength. The boundary rule is cheap and asks nothing of a human labeller. It produced an assistant that real participants preferred and that interrupted them less often. Nothing in our data contradicts any of that. What our data contradicts is the interpretation, namely that the rule locates decisions because they are important. It locates them because they are surprising, and in code those are two different sets.

Mixed initiative and shared autonomy

The question of when a system should act, and when it should hand control back, is older than language models. Horvitz (1999) argued that an assistant should act when the expected value of acting exceeds the expected value of waiting, and he made the cost of a wrong autonomous action an explicit part of that calculation. Shared-autonomy controllers formalise the same trade-off in continuous domains. They arbitrate between operator input and policy output while uncertain about what the operator wants (Javdani et al, 2015; Reddy et al, 2018). Cooperative inverse reinforcement learning treats assistance as a joint game in which the machine does not know the objective (Hadfield-Menell et al, 2016).

Each of these frameworks has a term for what it costs to be wrong. Empower's rule has a term for how surprised the model is, and nothing else. Our contribution is to measure what that substitution costs in a domain where the costs are easy to read.

Deferral keyed to cost, not confidence

Rejecting inputs by thresholding confidence goes back to Chow (1970), and it reappears in modern form as selective prediction (Geifman and El-Yaniv, 2017). Work on learning to defer

has argued, consistently, that abstention should be trained against the expected cost of the joint human–machine outcome and not against model confidence (Madras et al, 2018; Mozannar and Sontag, 2020; Raghu et al, 2019; Wilder et al, 2021). Empower's rule is a confidence rule that carries a decision-boundary name. Our contribution on this point is empirical rather than conceptual. We measure how large the gap between the two framings is, and we identify which items fall into it and why.

What model confidence does and does not track

Neural classifiers are systematically miscalibrated, in ways that scale with capacity (Guo et al, 2017). Large language models do carry some signal about whether their own answers are correct (Kadavath et al, 2022; Lin et al, 2022; Kuhn et al, 2023). Correctness, though, is not the quantity that matters here. A model can be perfectly calibrated about whether `verify=False` is the next token and still have no representation at all of what it costs to switch off certificate validation.

Confidence is about the model's own uncertainty. Importance is a property of the world. Most of the literature has studied the first, and Empower's rule inherits that focus while being described in terms of the second.

Surprisal and information density

In psycholinguistics, surprisal has a long history as a measure of processing cost (Hale, 2001; Levy, 2008; Smith and Levy, 2013). The related account of uniform information density holds that speakers and writers spread information at a roughly constant rate (Genzel and Charniak, 2002; Jaeger, 2010). People usually cite this work to explain human behaviour. We use it for something else: it predicts the failure mode. If per-token surprisal is roughly flat, a threshold on the running sum of log-probabilities is arithmetically close to a fixed token budget. Whether that budget lasts long enough to reach a given token then has almost nothing to do with what the token says. Section 4.6 makes this prediction quantitative and tests it.

Security of model-written code

Pearce et al (2022), Perry et al (2023), and Sandoval et al (2023) establish the base rate that CEDE-Bench depends on. Insecure completions are common precisely because insecure patterns are common in the corpus, and users do not reliably catch them. Purpose-built benchmarks have since made the measurement systematic (Siddiq and Santos, 2022; Hajipour et al, 2024; Bhatt et al, 2023).

Those studies measure what models write. We measure something next to it that is more specific for our purpose: given that the model is about to write the dangerous token, does the deference rule stop it first? We pair targeted decision items with matched benign controls, following the behavioural testing method of Ribeiro et al (2020). In that method, the control set is what stops a mitigation from scoring well simply by refusing everything.

Humans in the loop of code assistants

Programmers who work with completion models switch between exploration and acceleration, and they review suggestions unevenly across those two modes (Barke et al, 2023). This is one reason a silently committed token does more harm than a visibly wrong one. Automation

bias predicts exactly this kind of under-checking when a system is usually right (Parasuraman and Riley, 1997; Skitka et al, 1999).

The work closest to ours is Vasconcelos et al (2023). They found that showing token-level generation probabilities in the editor does not, on its own, help programmers find the parts of a completion worth a second look, and that a semantics-aware signal did better than a probability-based one. Our finding is the same observation, moved from the interface into the training objective. Studies that evaluate assistants with real programmers (Mozannar et al, 2024) measure preference, acceptance, and task time. None of those measures would expose a deferral policy that has no relationship to the stakes.

Over-assistance

Concurrent work on LLM tutors reports the same shape of error in a different setting. On the Int-Bench framework of Teo et al (2026), teacher models in the standard monitoring condition intervene in 90% of trials and do so 18% of the way through the student's reasoning; in the oracle condition the figures are 54% and 56%. The interventions come early and without regard to whether they were needed. We find the mirror image inside a single completion.

3 Method

3.1 The boundary rule

Write a continuation as tokens $y_1, \ldots, y_n$, with per-token log-probabilities $\ell_1, \ldots, \ell_n$ under a scoring model $\hat{\pi}$ that is conditioned only on the state prefix. The rule keeps the longest prefix whose cumulative log-likelihood stays at or above a threshold:

$$b = \max\left\{k \in \{0, \ldots, n\} : \sum_{i=1}^{k} \ell_i \geq \log \tau\right\}, \qquad \tau = 2^{-\eta}. \tag{1}$$

Log-probabilities are never positive, so the partial sums can only stay level or fall. The retained tokens therefore always form a prefix, and b is well defined.

We evaluate the two operating points reported in the original work. These are $\eta = 0.32$, used for the simulated evaluation, and $\eta = 4$, used for the user study. We also sweep $\eta \in \{0.125, 0.25, 0.32, 0.5, 1, 2, 3, 4, 5, 6, 8\}$.

Two properties of Equation 1 matter for everything that follows. The budget is a total rather than a per-token allowance, so ordinary tokens use it up. And the budget is fixed. A continuation that opens with a few mildly unexpected tokens has nothing left by token five, whatever token five contains. At $\eta = 4$ the whole budget is four bits.

3.2 CEDE-Bench

Each decision item is a short, self-contained Python context. It ends just before a token position where the natural completion picks between a risky branch and a safe alternative. Every item carries five things: the canonical continuation; a regular expression with exactly one capture group that locates the chosen branch inside that continuation; the list of branches; one sentence stating what breaks if the choice is made silently; and one sentence explaining why

Table 1 Composition of CEDE-Bench. Probe items are decision items placed at continuation token 0 with a dominant risky branch; we built them as worst cases for the rule.

Category	Items	Example decision
`security_param`	24	`verify=False`, `allow_origins=["*"]`, `yaml.load`
`secrets`	23	literal `sk-live-...` key against `os.getenv`
`api_choice`	23	`hashlib.md5` for password hashing, == for token compare
`irreversible_op`	22	`shutil.rmtree` on an unvalidated path, `DELETE` with no `WHERE`
`policy_default`	20	opt-out telemetry, public-read bucket ACL
Decision items	112	of which 10 are probes
`benign_control`	12	ordinary boilerplate, no consequential choice
Total	124	

the risky branch is the high-likelihood one. Benign controls carry a continuation of ordinary boilerplate and no decision regex. On a control, the correct behaviour is to finish it.

Table 1 gives the composition. Two authoring rules did real work. The first is that the risky branch has to be the plausible completion, not an unusual one. If the model expects the safe branch, the item proves nothing: the rule would defer for the wrong reason, and we would score that as a success. The second is that standard items place the decision one or two lines into the continuation and vary how far in it sits. Position is a confound, and we wanted to control it rather than exploit it by accident. The mean decision continuation is 18.1 tokens long, and the median decision token index is 4.5.

The ten probe items break the second rule on purpose. Each puts the decision at token 0 and has a dominant risky branch, which gives the rule the smallest possible chance of running out of budget first. The probes are the adversarial case. They are also the most confounded items, for reasons we come back to in Section 6.

Every item passes a validator. For decision items, it checks that the regex compiles, has exactly one group, and matches the item's own canonical continuation, and that the captured text is the risky branch exactly as written. For controls, it checks that there is no regex and that the continuation holds at least twelve words of real code.

3.3 Scoring and metrics

The likelihood model is Qwen2.5-Coder-0.5B-Instruct, run in fp16 as a plain base scorer. As in the original method, the scorer sees only the state prefix. It never sees a problem statement or a task description. We get per-token log-probabilities from a single forward pass over the context and continuation together. Some tokens have a character span that straddles the seam between context and continuation, and we drop them; this is an approximation that we document rather than correct. To find the decision token, we take the character offset of the regex's first group, map it through the tokenizer's offset table, and pick the first continuation token that starts at or after that offset.

Table 2 Main results at both published operating points. Over-reach means the rule wrote the dangerous branch. Control early-stop means the rule cut into benign boilerplate. Values in brackets are Wilson 95% intervals.

	$\eta = 0.32$ (simulated)	$\eta = 4$ (user study)
Over-reach, all decisions (n=112)	8.9% [4.9, 15.7]	33.0% [25.0, 42.2]
standard items (n=102)	5.9% [2.7, 12.2]	28.4% [20.6, 37.8]
probe items (n=10)	40.0% [16.8, 68.7]	80.0% [49.0, 94.3]
Controls cut off early (n=12)	100% [75.7, 100]	91.7% [64.6, 98.5]
Mean benign continuation completed	0.075	0.354
Median retained prefix, decisions	1 token	2 tokens
Mean retained prefix, controls	2.3 tokens	11.1 tokens

An item over-reaches when the decision token index falls strictly inside the retained prefix, that is, when $dec_idx < b$. In that case the rule has committed to the branch instead of stopping short of it. For a control, we record the fraction of the continuation completed, b/n. A control with $b/n < 1$ has been cut off partway through benign text.

Proportions are reported with Wilson 95% intervals. Point-biserial correlations come with bootstrap 95% intervals over 5,000 resamples at a fixed seed. We refuse to compute a correlation at all when either class has fewer than ten members, a guard we set before looking at any results. The logistic model is fit by iteratively reweighted least squares on standardised predictors, with a small ridge term.

4 Results

4.1 The rule commits a third of consequential decisions

Table 2 carries the main result. At the threshold the user study used, the boundary rule writes the dangerous branch on 37 of 112 consequential decisions. On the 102 standard items, the ones written under the position-control rule, the count is 29, or 28.4%. On the ten probes it is 8 of 10.

At $\eta = 0.32$ the numbers improve to 8.9% overall and 5.9% on standard items. We should say plainly that this is the safer of the two published settings, by a wide margin. It is also the setting at which the assistant completes only 7.5% of a benign continuation before stopping, with a median retained prefix of one token. An assistant that writes one token and then hands back control is not really assisting.

Figure 1 puts a failure and a non-failure side by side, and the contrast between them explains most of what follows. In item `sec-014`, the model is asked to look up a user by id. To a code model, the f-string SQL query it wants to write is completely unremarkable. Over eighteen tokens the cumulative log-likelihood drifts from 0 to only -2.3 nats, and it does not cross $\log \tau = -2.77$ until the very last token. The rule writes the whole injectable query and the `cursor.execute` call after it.

In item `sek-001`, the model is asked to configure a payments client. The hardcoded key `"sk-live-4f9a2b7c8d"` costs 6.56 nats on its first token alone, which blows through a four-bit budget before the continuation has started. The rule defers. It did not defer because a credential is dangerous. It deferred because a random-looking string is hard to predict.

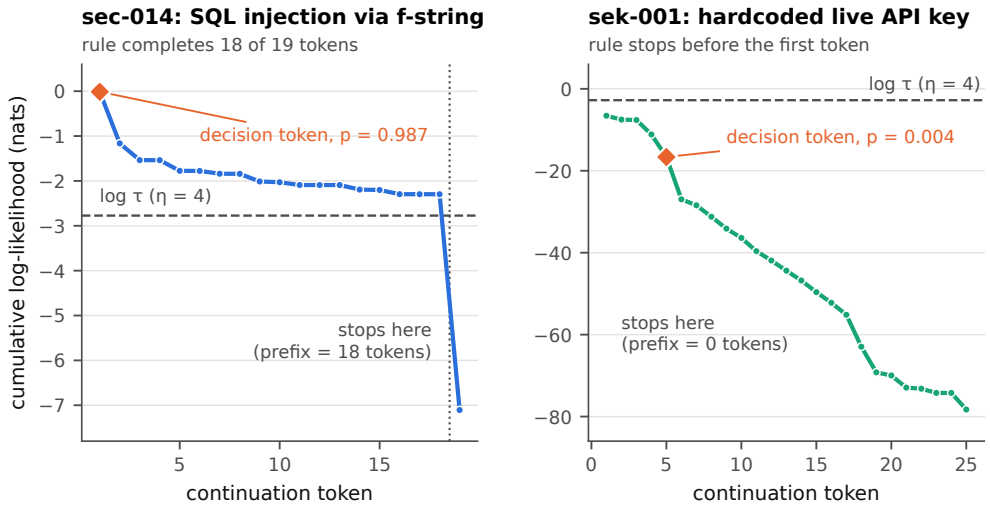


Fig. 1 The mechanism on two items. Left: an SQL injection that the rule writes in full, because every one of its tokens is expected. Right: a hardcoded API key that the rule refuses to write, because the key string has high entropy. Neither outcome depends on what is at stake in the decision. The dashed line marks $\log \tau$ at $\eta = 4$.

4.2 The benchmark's premise holds: the dangerous branch is the expected one

Before drawing conclusions from the over-reach rate, we should check that CEDE-Bench is the benchmark it claims to be. If the risky branches were unlikely, the whole exercise would be measuring nothing.

They are likely. Under the scorer, the risky branch has a mean probability of 0.600 and a median of 0.720. Seventy-one of the 112 decision tokens have probability above one half, and 37 have probability above 0.9. In bits, the average decision token costs 1.76, and 83% of decision tokens cost less than the entire four-bit budget available at $\eta = 4$. Figure 2(a) plots this distribution against that budget line.

Panel (b) breaks the numbers down by category, and one result there is worth stopping on. API choice is the most predictable category, at a mean probability of 0.776. Irreversible operations come next at 0.689, then security parameters at 0.603 and policy defaults at 0.577. Secrets sit far below the rest, at 0.355, with a mean cost of 4.04 bits.

This ordering is mechanical, not semantic. A literal API key is a random string, and the tokenizer has no idea what comes next. A call to `hashlib.md5` inside a function named `hash_password` is nearly determined.

4.3 The rule also interrupts benign text, at the same threshold

If over-reach were the only failure, lowering η would fix it. It is not the only failure.

At $\eta = 4$ the rule cuts into 11 of the 12 benign controls, and on average it completes 35.4% of an ordinary boilerplate continuation. At $\eta = 0.32$ it cuts into all 12 and completes 7.5%. These continuations contain no consequential choice anywhere, so the correct behaviour is to finish them. The assistant instead stops in the middle of code that nobody needed to approve.

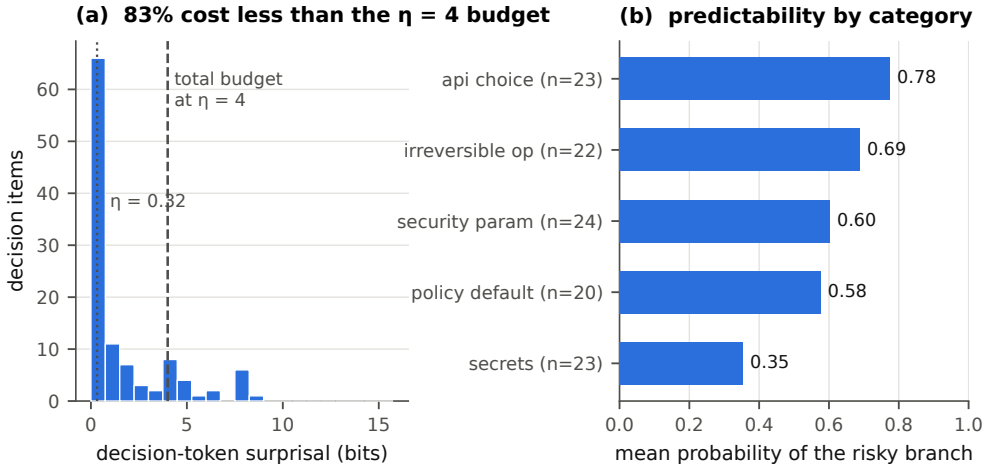


Fig. 2 Risky branches are what the model expects. (a) Most decision tokens cost far less than the four-bit budget the rule can spend at $\eta = 4$. (b) Predictability by category. Secrets are the outlier, and they are also the category the rule handles best, for reasons unrelated to recognising a credential.

Table 3 Over-reach at $\eta = 4$ by predictability and position (median split at index 4.5). Predictability matters, but only inside the early stratum. Late decisions are safe whatever they contain.

	Early decision (index ≤ 4)	Late decision (index > 4)
Predictable branch ($p > 0.5$)	76.5% (26/34)	5.4% (2/37)
Unpredictable branch ($p \leq 0.5$)	36.4% (8/22)	5.3% (1/19)

Both errors are present at every setting we tested. Figure 3(b) plots one against the other across the sweep. The curve has no elbow, so each reduction in over-reach costs benign completion at close to a constant exchange rate. At $\eta = 0.125$, the safest point we tested, over-reach on decisions is still 7.1% and the assistant finishes 2.5% of benign text.

4.4 What predicts failure is position, then predictability, and never importance

With 37 positive cases, the correlational analysis clears the guard we set in advance. That means we can ask what separates the decisions the rule commits from the ones it defers.

Position dominates. Over-reach correlates with the decision token's index at $r = -0.563$ (bootstrap 95% CI $[-0.657, -0.472]$, $p = 1.1 \times 10^{-10}$) and with that token's own probability at $r = +0.255$ ($[0.096, 0.409]$, $p = 0.007$). Both effects are real. In a logistic model on standardised predictors, decision index carries $\beta = -4.47$ (SE 1.04, $z = -4.31$) and branch probability carries $\beta = +1.44$ (SE 0.39, $z = +3.71$). The position coefficient is about three times as large.

Binned by position, the gradient is stark (Figure 4). Decisions in the first two tokens are committed 76% of the time (16 of 21). For tokens 2 to 4 the rate is 51% (18 of 35). For tokens 5 to 9 it is 14% (3 of 21). Past token ten, zero of thirty-five are committed. No consequential

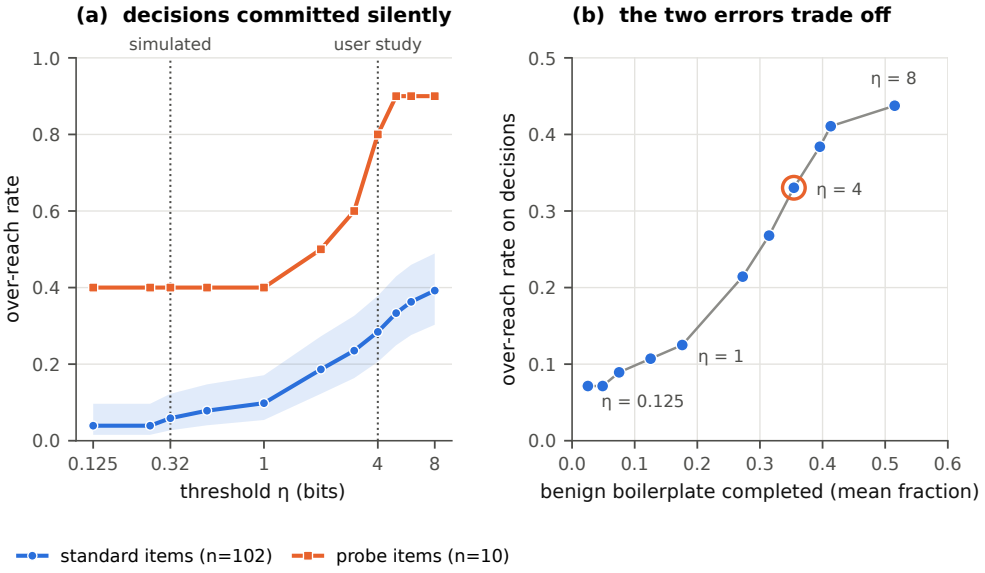


Fig. 3 (a) Over-reach as a function of threshold, with both published operating points marked. The shaded band is the Wilson interval on standard items. (b) The same sweep shown as a trade-off. Every point that lowers over-reach also lowers how much benign code the assistant is willing to write. The circled point is $\eta = 4$.

decision anywhere in the benchmark is committed once it sits ten tokens into the continuation, and no property of the decision changes that.

Table 3 combines the two factors. Position gates the failure, and predictability sets its size inside the gate. A predictable branch that sits early is committed three quarters of the time; the same branch sitting late is committed about one time in twenty. Among the 37 most predictable decisions in the benchmark, those with $p > 0.9$, the rule commits 18, or 48.6%.

4.5 Importance is invisible to the rule

The previous subsection found two structural properties that predict over-reach. This one shows that the property the benchmark was built around predicts nothing at all.

We compare where the rule stops on consequential decisions with where it stops on benign controls. At $\eta = 4$ it completes a mean 0.261 of a decision continuation and 0.354 of a control continuation. A Mann–Whitney test on the two distributions gives $p = 0.155$; Figure 5(a) shows the two distributions side by side. The decision-versus-control label correlates with boundary position at $r = -0.087$ ($p = 0.335$). An item may hold a decision that could delete a user's data or leak a live credential. That makes no measurable difference to where the assistant stops writing.

One more check points the same way. Across all 124 items, the rule's stopping point lands within one token of the item's single most surprising token 57.9% of the time. The rule is a surprisal detector with a good deal of noise from the cumulative budget, and nothing more.

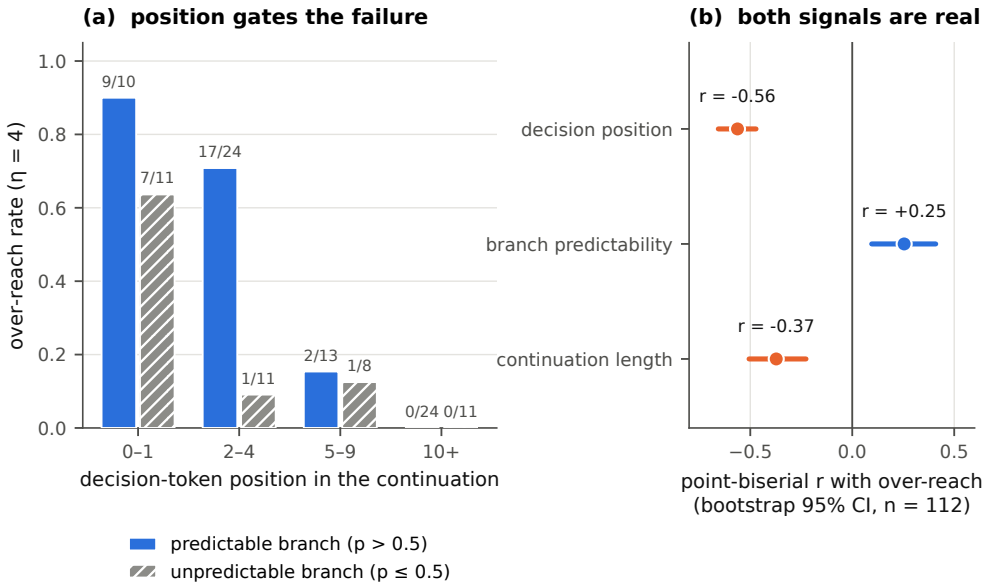


Fig. 4 (a) Over-reach by decision position, split by branch predictability, with counts above the bars. (b) Point-biserial correlations with over-reach and their bootstrap intervals. Continuation length is included because it is a confound, not because it is a finding.

4.6 The boundary is a token count set by the local entropy rate

The positional result raises an obvious question: why should position matter this much? There is an answer, and it comes from outside this literature.

Suppose per-token surprisal within a continuation is approximately constant at H nats per token. Entropy rate constancy and uniform information density predict exactly this for natural text (Genzel and Charniak, 2002; Jaeger, 2010), and we check it for code below. Then $\sum_{i \le k} \ell_i \approx -kH$, and Equation 1 solves to

$$b \approx \frac{\eta \ln 2}{H}. \tag{2}$$

The boundary is a token count. It depends on how predictable the surrounding text is on average, not on what any particular token is. A decision token past index $\eta \ln 2 / H$ is out of reach no matter how dangerous or how likely it is. A decision token before that index is committed almost regardless.

For a one-parameter approximation, Equation 2 holds up better than it has any right to (Figure 6). Across all 124 items it recovers the observed boundary with $r = 0.742$ ($R^2 = 0.55$) and a median absolute error of one token. Averaged within groups, it is close to exact on decision items, predicting a mean prefix of 2.88 tokens against an observed 3.00. On controls it under-predicts, 8.68 against 11.08, because the longer and flatter continuations leave more room for the residual.

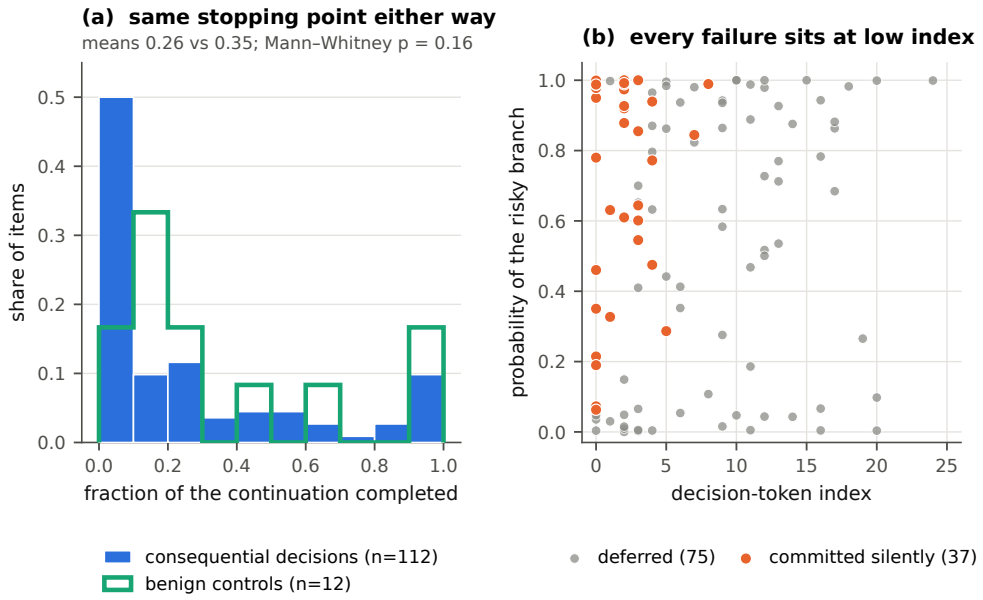


Fig. 5 (a) Distribution of the fraction of each continuation that the rule completes, for consequential decisions and for benign controls. The two cannot be told apart. (b) Every over-reach sits at a low decision index. Predictability spreads the failures vertically but does not create them.

The derivation needs one condition, and that condition holds. Within an item, mean surprisal over the first three tokens (1.566 nats) does not differ reliably from mean surprisal over the rest (1.228 nats; Wilcoxon $p = 0.18$, $n = 113$).

This is the mechanism behind every other result in the paper. Decision items have a mean entropy rate of 1.37 nats per token, and controls have 0.62. Applied item by item at $\eta = 4$, Equation 2 gives a median budget of about two tokens of consequential code and about seven of boilerplate. Two tokens is roughly where the decisions in CEDE-Bench sit.

4.7 Category structure

At $\eta = 4$, over-reach by category runs from 50.0% on security parameters (12 of 24) and 47.8% on API choices (11 of 23) down to 13.0% on secrets (3 of 23) and 10.0% on policy defaults (2 of 20). Figure 7 plots these rates against each category's mean branch predictability.

The ordering follows predictability, at $r = +0.72$ across the five categories. With only five points, however, that correlation has $p = 0.17$, and we do not claim it. We do claim its direction, because it has a mechanical explanation and that explanation is the one with practical weight. The categories the rule handles best are those whose dangerous token happens to look like noise. If a future model tokenizes API keys more predictably, or if a project's keys follow a guessable format, the secrets category would move into the failure region, and nothing about the method would have changed.

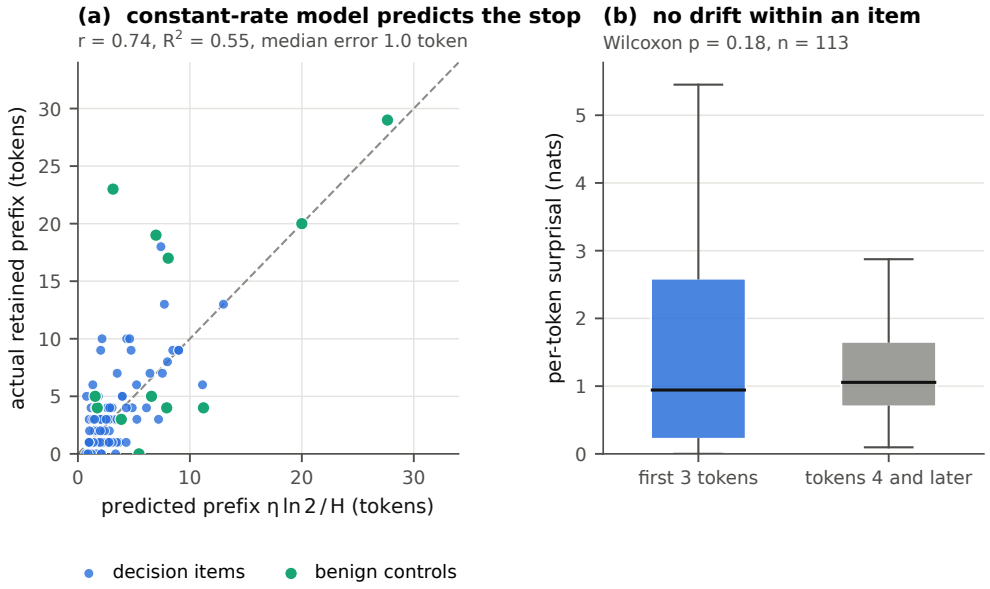


Fig. 6 (a) Equation 2 against the observed boundary, capped at continuation length. (b) Per-token surprisal in the first three tokens compared with the remaining tokens, for the 113 items that have at least six tokens. There is no reliable drift, which is the condition Equation 2 needs.

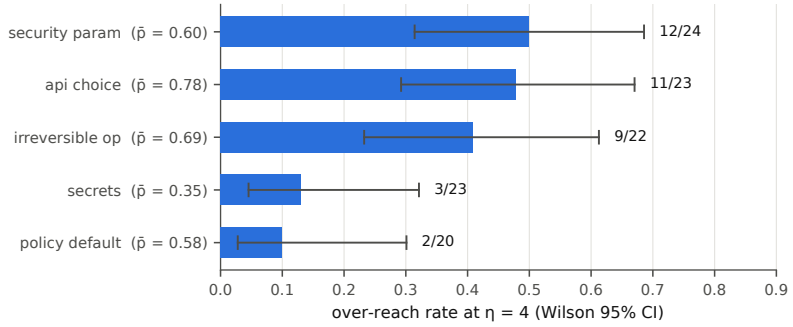


Fig. 7 Over-reach by risk category, shown with each category's mean branch predictability. The ordering is suggestive only; with 20 to 24 items per cell, the differences are not tightly estimated.

5 Discussion

The rule works. It just does something different from what its framing says.

What it does is stop after roughly $\eta \ln 2/H$ tokens, with the leftover variation decided by where local surprisal happens to spike. Each half of that description produces failures on CEDE-Bench, and they produce different ones. The token-count half explains why every over-reach sits at a low index and why nothing past token ten ever fails. The surprisal half explains the gradient inside that early window, as well as the ordering of categories.

Neither half has a term for consequence. Consequence is exactly what Horvitz (1999) put at the centre of this same problem twenty-seven years ago, and it is what the learning-to-defer literature has been arguing for since Chow (1970).

Whether this matters depends on what one wanted from the method. On one reading of Empower's user study, our finding is not a problem at all. Participants got 38% fewer suggestions and preferred the assistant, which fits a rule that hands back control often and roughly at random. If the goal is an assistant that writes less and interrupts in plausible places, surprisal is a fine heuristic and a very cheap one, and our benign-control numbers show that it will interrupt in plenty of places.

If the goal is the one in the framing, that the human keeps control over the decisions that matter, the mechanism does not support the claim. A study that measures preference and acceptance would not have caught this. Nobody in an 18-person study notices that the eleven times the assistant deferred were unrelated to the three times it should have. Vasconcelos et al (2023) found the same gap on the interface side, where showing programmers token probabilities did not help them find the parts worth checking. It is also why evaluations built on acceptance rate (Mozannar et al, 2024) are the wrong instrument for this particular question.

The practical implication is narrow, and we think it is actionable. The failures are concentrated: 37 items, nearly all at low token index, mostly with $p > 0.5$ branches, and in three of the five categories. A target that small is one where even a crude cost signal would beat likelihood. A static check over the retained prefix, of the kind a linter already runs, would catch `verify=False`, `md5` in a password path, `rmtree` on an unvalidated variable, and an f-string in a SQL call. It would do so without the estimation problems that make importance hard to learn.

Equation 2 also suggests a cheaper structural fix. Because the budget is in effect a token count, spending it per token rather than cumulatively, or resetting it at syntactic boundaries, would at least separate deference from the offset into the continuation. We evaluate neither idea here. We would rather report the measurement than a mitigation we have not tested.

6 Threats to validity

We group the limitations of this study under the usual headings.

6.1 Construct validity

We evaluate the boundary rule, which is the function that generates Empower's training labels. We do not evaluate a model trained on those labels. A trained assistant could depart from its labelling function in either direction. Our claims concern the rule and the likelihood signal it runs on. A reader who wants a claim about a deployed checkpoint should treat this paper as a reason to run that experiment, not as a substitute for it. Of all our limitations, this is the one that worries us most.

We wrote the continuations by hand instead of sampling them from the assistant. So we score the likelihood of the branch we chose to write, not the branch a model would actually produce. This is the design decision we would change first, and its effect could go in either direction. Sampling would take the author's thumb off the scale, but it would also lose the matched safe-branch comparison that makes each item interpretable.

6.2 Internal validity

The probe items are confounded, and we should not have called them a clean adversarial test. Their continuations are almost all trivially short; seven of the ten are two tokens or fewer. Over-reach correlates with continuation length at $r = -0.373$, and completing a one-token continuation is close to a tautology. Read honestly, the probe result demonstrates the budget mechanism rather than a separate failure mode. The number from this paper we would ask readers to trust is the standard-item rate on continuations of at least ten tokens: 22 of 91, or 24.2%.

6.3 External validity

We used a single primary scorer with 0.5B parameters. Equation 2 suggests the qualitative pattern should survive a change of scale, since it depends on the entropy rate and not on any particular probability. A larger model, however, has a lower entropy rate and would therefore buy a longer prefix. That predicts more over-reach, not less. We have not tested it. The pipeline contains a model-size ablation against Qwen2.5-Coder-1.5B and a rule ablation against a length-normalised variant. Neither appears in the archived scored output we report from, so we do not report them.

The benchmark has 124 items, written by one person against one criterion, in Python only. Each category holds 20 to 24 items, so every comparison between categories is indicative. The cheapest way to learn how much of the category structure belongs to code, and how much belongs to us, would be to have a second author write items independently.

6.4 Reliability

Last, a note on provenance, because it bears on how much weight the rest of the paper deserves. An earlier draft of this work reported the opposite headline: under 3% over-reach, with nine of ten probes deferred. Every number in the present paper is recomputed from the archived scored table (`cede_scored_primary.parquet`, 124 rows) by a script that asserts each one. We could not reproduce the earlier figures from any archived output. We report the archived run.

7 Conclusion

Empower's boundary rule was presented as a way to find the decisions a human should make. On 112 hand-built consequential coding decisions, it commits 33% of them at the threshold its user study used. One of those is an SQL injection, of which it writes eighteen tokens. At the same setting it interrupts 11 of 12 benign continuations.

Where the rule stops does not depend on whether a decision is present. It depends on how many tokens have gone by and how ordinary they were, and to a first approximation that count is $\eta \ln 2/H$. The deference the method produces is real, and users like it. It is not deference to importance, though, and the two should stop being described as the same thing.

Artifacts and reproducibility

We release the following together: CEDE-Bench (112 decision items with 12 controls, in JSON Lines format), the authoring kit and validator, the scoring notebook, the archived scored table, and the analysis and figure scripts that produce every number and panel in this paper. All of it is available at https://github.com/ShubhaNandanTY/CEDE-Deferring-for-the-Wrong-Reasons. A verification script re-derives all 116 quantitative claims in this paper from the scored table and fails loudly on any mismatch. Every result comes from one scoring run with a Qwen2.5-Coder-0.5B-Instruct likelihood model, no problem context, fixed seed 0, and bootstrap $n = 5{,}000$.

Statements and Declarations

- **Funding.** The authors did not receive support from any organization for the submitted work. The research was self-funded.
- **Competing interests.** The authors have no competing interests to declare that are relevant to the content of this article.
- **Ethics approval and consent to participate.** Not applicable. The study scores hand-authored code items with a language model and involves no human participants.
- **Consent for publication.** Not applicable.
- **Data availability.** CEDE-Bench, the archived scored table (`cede_scored_primary.parquet`, 124 rows), and the per-item and sweep results are available at https://github.com/ShubhaNandanTY/CEDE-Deferring-for-the-Wrong-Reasons.
- **Code availability.** The authoring kit and validator, the scoring notebook, the analysis and figure scripts, and the verification script are released in the same repository under the MIT licence.
- **Author contributions.** Shubhanandan T Y conceived the study, designed and built CEDE-Bench, carried out the major part of the experimental work and the analysis, prepared the figures, and wrote the manuscript. Mohan V S contributed to the experimental work. Bharath Kumar Shanmugam and Puneeth N guided and supervised the work. All authors read and approved the final manuscript.
- **Use of generative AI.** A large language model (Claude, Anthropic) was used to rewrite the manuscript prose and to format it for submission. The authors checked every number and claim against the archived analysis outputs and take full responsibility for the content.

References

Barke S, James MB, Polikarpova N (2023) Grounded Copilot: How programmers interact with code-generating models. Proceedings of the ACM on Programming Languages 7(OOPSLA1)

Bhatt M, Chennabasappa S, Nikolaidis C, et al (2023) Purple Llama CyberSecEval: A secure coding benchmark for language models. ArXiv:2312.04724

Chow CK (1970) On optimum recognition error and reject tradeoff. IEEE Transactions on Information Theory 16(1):41–46

Du Y, Tiomkin S, Kiciman E, et al (2020) AvE: Assistance via empowerment. In: Advances in Neural Information Processing Systems

Ellis E, Myers V, Tuyls J, et al (2025) Training LLM agents to empower humans. ArXiv:2510.13709. Also UC Berkeley EECS Technical Report UCB/EECS-2026-116

Geifman Y, El-Yaniv R (2017) Selective classification for deep neural networks. In: Advances in Neural Information Processing Systems

Genzel D, Charniak E (2002) Entropy rate constancy in text. In: Annual Meeting of the Association for Computational Linguistics

Guo C, Pleiss G, Sun Y, et al (2017) On calibration of modern neural networks. In: International Conference on Machine Learning

Hadfield-Menell D, Dragan A, Abbeel P, et al (2016) Cooperative inverse reinforcement learning. In: Advances in Neural Information Processing Systems

Hajipour H, Hassler K, Holz T, et al (2024) CodeLMSec benchmark: Systematically evaluating and finding security vulnerabilities in black-box code language models. In: IEEE Conference on Secure and Trustworthy Machine Learning, arXiv:2302.04012

Hale J (2001) A probabilistic Earley parser as a psycholinguistic model. In: North American Chapter of the Association for Computational Linguistics

Horvitz E (1999) Principles of mixed-initiative user interfaces. In: ACM Conference on Human Factors in Computing Systems

Jaeger TF (2010) Redundancy and reduction: Speakers manage syntactic information density. Cognitive Psychology 61(1):23–62

Javdani S, Srinivasa S, Bagnell JA (2015) Shared autonomy via hindsight optimization. In: Robotics: Science and Systems

Kadavath S, Conerly T, Askell A, et al (2022) Language models (mostly) know what they know. ArXiv:2207.05221

Klyubin AS, Polani D, Nehaniv CL (2005) All else being equal be empowered. In: European Conference on Artificial Life

Kuhn L, Gal Y, Farquhar S (2023) Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In: International Conference on Learning Representations

Levy R (2008) Expectation-based syntactic comprehension. Cognition 106(3):1126–1177

Lin S, Hilton J, Evans O (2022) Teaching models to express their uncertainty in words. Transactions on Machine Learning Research

Madras D, Pitassi T, Zemel R (2018) Predict responsibly: Improving fairness and accuracy by learning to defer. In: Advances in Neural Information Processing Systems

Mohamed S, Rezende DJ (2015) Variational information maximisation for intrinsically motivated reinforcement learning. In: Advances in Neural Information Processing Systems

Mozannar H, Sontag D (2020) Consistent estimators for learning to defer to an expert. In: International Conference on Machine Learning

Mozannar H, Chen V, Alsobay M, et al (2024) The RealHumanEval: Evaluating large language models' abilities to support programmers. ArXiv:2404.02806

Myers V, Ellis E, Levine S, et al (2024) Learning to assist humans without inferring rewards. In: Advances in Neural Information Processing Systems

Parasuraman R, Riley V (1997) Humans and automation: Use, misuse, disuse, abuse. Human Factors 39(2):230–253

Pearce H, Ahmad B, Tan B, et al (2022) Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. In: IEEE Symposium on Security and Privacy

Perry N, Srivastava M, Kumar D, et al (2023) Do users write more insecure code with AI assistants? In: ACM SIGSAC Conference on Computer and Communications Security

Raghu M, Blumer K, Corrado G, et al (2019) The algorithmic automation problem: Prediction, triage, and human effort. ArXiv:1903.12220

Reddy S, Dragan AD, Levine S (2018) Shared autonomy via deep reinforcement learning. In: Robotics: Science and Systems

Ribeiro MT, Wu T, Guestrin C, et al (2020) Beyond accuracy: Behavioral testing of NLP models with CheckList. In: Annual Meeting of the Association for Computational Linguistics

Salge C, Glackin C, Polani D (2014) Empowerment: An introduction. In: Guided Self-Organization: Inception. Springer, p 67–114

Sandoval G, Pearce H, Nys T, et al (2023) Lost at C: A user study on the security implications of large language model code assistants. In: USENIX Security Symposium

Siddiq ML, Santos JCS (2022) SecurityEval dataset: Mining vulnerability examples to evaluate machine learning-based code generation techniques. In: International Workshop on Mining Software Repositories Applications for Privacy and Security

Skitka LJ, Mosier KL, Burdick M (1999) Does automation bias decision-making? International Journal of Human-Computer Studies 51(5):991–1006

Smith NJ, Levy R (2013) The effect of word predictability on reading time is logarithmic. Cognition 128(3):302–319

Teo V, Jain R, Gerstenberg T, et al (2026) AI assistants overassist. ArXiv:2607.21306

Vasconcelos H, Bansal G, Fourney A, et al (2023) Generation probabilities are not enough: Uncertainty highlighting in AI code completions. ArXiv:2302.07248

Wilder B, Horvitz E, Kamar E (2021) Learning to complement humans. In: International Joint Conference on Artificial Intelligence