

Letting AI Agents Choose Without Letting Them Whisper: Proved Leakage Caps Against Adaptive LLM Colluders

Teo Asinari
Independent researcher
tasinari@protonmail.com

Apart Research AI Collusion Research Sprint, October 23–25, 2026

Abstract

AI agents that work together can hide secret messages inside ordinary answers, for example in wording, formatting, or which of several equally good answers they pick. We study defenses that limit this. First, rewriting an agent’s message into a standard form removes every hidden signal we tried except one: which answer the agent chose. Second, we restrict that choice to a menu. A trusted model writes a short list of good answers and the agent may only pick one. Picking one of 8 options carries at most 3 bits, so at most 3 secret bits can leak per answer, and we prove this limit in Lean for every possible model. Third, networks built to depend only on approved information leak nothing, but leave the agent no freedom. Finally, we tested the defenses against real AI agents (GPT-4.1-mini, coached by Claude Sonnet 4.6) that improved their own secret code over 12 rounds, in five independent runs. Unprotected, they leaked up to 7 of 8 secret bits per answer. With menus of 2, 4, 8, 16 and 32 options, they leaked 1.0, 1.9, 2.9, 3.7 and 4.5 bits, against proven limits of 1 to 5, never meaningfully above them. Larger menus also gave better honest answers than free answers. The standard-form defense leaked nothing in three runs but about 2 bits in two, once the agents found its loophole.

1 Introduction

Multi-agent LLM systems give agents many ways to communicate covertly: wording, formatting, ordering, tool parameters and the substantive choices agents make. Recent work shows tool-using agents can implement provably secure stegosystems against output-only monitors [4]. Such schemes build on perfectly secure constructions like minimum-entropy coupling [7]. Optimization against oversight can also produce encodings that survive paraphrasing [3]. On the defensive side, paraphrasing reduces the capacity of tested encodings but does not eliminate it [5]. Trusted-prefix interventions reduce measured capacity per decision point [2].

Rewriting-based defenses rest on a tension. A defender can remove any information it is willing to regenerate. But a useful agent is useful because its choices depend on information the defender lacks. Whatever the defender preserves of those choices is available as a channel. We make this tension concrete in two steps:

1. **Erasure (§3).** A trusted broker for a structured task erases all presentation-level carriers we tested. The semantic carrier, i.e. which of several equally good solutions is chosen, survives every mode that preserves the agent’s decision.
2. **Bounding (§4).** Instead of erasing the residual choice, we restrict it. The untrusted model may only select one of 2^k candidates produced by a trusted public process. We prove in Lean

that this caps leakage at k bits per answer for every weight setting. We then measure how much task utility k bits buy.

3. **Zero leakage (§5).** Learned residue networks whose released output provably depends only on authorized information occupy the $k = 0$ end: no covert channel, and no private freedom.
4. **Adaptive LLM colluders (§6).** We let GPT-4.1-mini colluders improve their own scheme over 12 rounds against eight defenses, including menus of 2–32 options, in five independent runs.

We distinguish channel *capacity* from instructed use, independent discovery and harmful coordination. Experiments 1–3 concern capacity under scripted encoders. Experiment 4 measures instructed colluders who adapt their scheme in context; it does not test spontaneous, uninstructed collusion.

2 Related work

Steganographic collusion. Rippin et al. [4] show tool-using agents implementing stegosystems undetectable by an output-only monitor, given a keyed cooperating receiver and access to sampling probabilities. Schroeder de Witt et al. [7] give perfectly secure steganography via minimum-entropy coupling. Mathew et al. [3] show steganography emerging under optimization pressure and evaluate paraphrasing mitigations.

Rewriting defenses. Roger and Greenblatt [5] evaluate paraphrasing against encoded reasoning. They report under 3 bits/KB for their strongest tested encodings and discuss the limits of replacing substantive content. Our broker experiment is a fully specified instance of this observation, in which the residual channel can be named exactly.

Capacity bounds by counting. Bounding leakage by the logarithm of the number of distinguishable observations is standard in quantitative information flow, e.g. predictive timing mitigation [1]. Choosing one of N items conveys at most $\log_2 N$ bits. *Hand and Brain* [2] is the closest agentic defense. A small trusted model writes the first tokens at each decision point of an untrusted model, reducing measured capacity from 2.13 to 0.08 bits per decision point on one task, without a proved bound. Our architecture inverts the roles: the untrusted model selects among trusted outputs. We do not claim the counting argument as new. The contribution is the typed architecture with a machine-checked bound valid for all weights, plus a measured utility-versus- k curve.

3 Experiment 1: what a content-erasing broker removes

Task and threat model. A sender agent submits a JSON message assigning workers to tasks under public integer costs. Before the receiver sees it, a trusted broker validates and transforms it. The sender wants to transmit a private bit Z . The receiver knows the encoding scheme and sees only the released message. The broker, the task specification P and the public cost oracle are trusted.

Table 1: Known-decoder accuracy (50% = chance) for each carrier under each release mode. All allocations are optimal by construction, so task utility is 100% in every non-replan cell. Replan changes half of the assignments under the semantic carrier.

Carrier	Identity	Sort JSON	Project	Rerender	Replan
Whitespace	100%	50%	50%	50.8%	50%
Object key order	100%	50%	50%	50%	50%
Assignment list order	100%	100%	50%	50%	50%
Escaped identifiers	100%	50%	50%	50%	50%
Optional wording	100%	100%	50%	50%	50%
Assignment choice	100%	100%	100%	100%	50%

Release modes. `identity` passes the validated message (positive control). `json_sort` re-serializes with sorted keys. `project` extracts only the validated assignment Q and renders it canonically. `rerender` renders the projected assignment with fresh defender-chosen formatting. `replan` ignores the message and computes an optimal assignment from P with a fixed tie break. The parser rejects duplicate keys, unknown identifiers, invalid assignments, nonfinite constants, invalid Unicode and oversized input.

Scoped guarantee. Projection releases $Y = f(P, Q)$, and rerendering releases $Y = g(P, Q, R)$ with defender randomness R independent of Z . Hence $I(Z; Y \mid P, Q) = 0$ for accepted messages. This follows from the construction, not from a statistical test. It deliberately conditions on Q : if the sender encodes Z in its choice of assignment, that information survives. Replanning releases $Y = h(P)$, which removes the dependence by removing the sender’s decision.

Carriers. Six scripted one-bit encoders: whitespace, object key order, assignment list order, escaped identifiers, optional free-text wording, and *actual assignment choice* between two equally optimal allocations. Each (carrier, mode) cell has 256 balanced trials, for 7,680 records in total.

Results. Table 1 shows that every positive control decodes perfectly. Syntactic normalization (`json_sort`) removes only some carriers. Projection and rerendering remove all presentation carriers. The semantic carrier survives every mode that preserves the sender’s assignment. Only replanning removes it, and replanning makes the sender irrelevant. Preserving task *utility* is weaker than preserving task *meaning*: equally optimal choices encode bits at no utility cost. Eight unit tests check the invariants, malformed-input rejection, reproducibility and the surviving semantic channel.

The defender therefore has two options. It can preserve the agent’s choices and leave a channel, or take over the choices and lose the agent’s contribution. Experiment 2 explores the space between these options: keep some of the agent’s choice, but bound it.

4 Experiment 2: a leak-capped selection architecture

Architecture. The receiver sees only $\text{DECODE}(x, r, c) = \text{CANDIDATES}(x, r)[c]$, which has three parts:

- A **public proposer** (trusted, public weights) samples 2^k candidate answers from the public input x and public randomness r . Candidate 0 is its greedy answer. It never sees private data.

Table 2: Leak-capped system with proposer 2 (105k). Unrestricted chooser: 29.7%. The 95% interval at $n = 600$ is about ± 3.7 points.

k	0	1	2	3	4	5	6	7	8
Accuracy (%)	15.5	16.5	18.5	20.3	24.0	29.2	32.7	34.0	35.0
Coverage (%)	15.5	16.7	20.0	23.3	31.8	43.2	51.7	59.3	66.8
Adversary bits	0	0.84	1.73	2.58	3.41	4.16	4.86	5.50	6.04

- An **untrusted chooser** reads x , all candidates and anything private (a payload, private weights or private context), and emits a k -bit code c . Honestly, c selects the candidate the chooser rates most likely.
- A **trusted decoder** outputs candidate c verbatim.

Work is fixed: the proposer always samples 2^k padded candidates and the chooser always scores all of them, so running time depends only on public k .

Formal guarantee. We formalize a small dataflow language in Lean 4.35.0. It has public operations, private operations (every function symbol arbitrary, so every network and weight setting is covered), **choose** nodes $out := dec(\text{public args}, c \bmod 2^k)$, and observations. Its typing rule labels each operation’s output with the join of its inputs’ labels. A **choose** output is public only if every decoder argument other than the code is public, and only public wires are observed. Theorem **view_count** states: for every interpretation and fixed public inputs, the receiver’s entire view lies in an explicit list of exactly $2^{k_1 + \dots + k_n}$ values, one per code sequence. Over all payloads, private inputs, private randomness and weights, the view therefore takes at most $2^{\sum k}$ values, and leakage is at most $\sum k$ bits (log-cardinality measure [1]). **seed_capacity** instantiates the proposer/chooser/decoder system as well-typed with at most 2^k views per answer. A necessity example shows that a decoder reading the private score directly is rejected, because that wiring leaks even at $k = 0$. The development has no **sorry** and uses only the standard axioms **propext** and **Quot.sound**. It re-checks in about 2s.

Toy task. The task is to denoise a corrupted common English word (lengths 4–10 from a 9,894-word list), where each letter is replaced with probability 0.3. All models are small transformers trained from scratch on CPU for 3,000 steps of batch 256: proposer 1 (15k parameters), proposer 2 (105k) and a chooser (605k). The baseline is the *same* chooser decoding freely without a cap. The test set has 600 fresh corruptions with public seed 7.

Adversary. A scripted adversary sets c to a uniform k -bit payload. The receiver recomputes the public candidates and decodes the index. We report exact mutual information $I(m; \text{answer} \mid x, r)$. We also run an adversary that tries to send $k + 1$ bits.

Results. Figure 1 and Table 2 summarize the results.

- The scripted adversary never exceeds k bits. It falls below k only because sampled candidates include duplicates, and trying to send $k + 1$ bits gains nothing.
- An unrestricted 10-letter answer could carry up to about 47 bits.

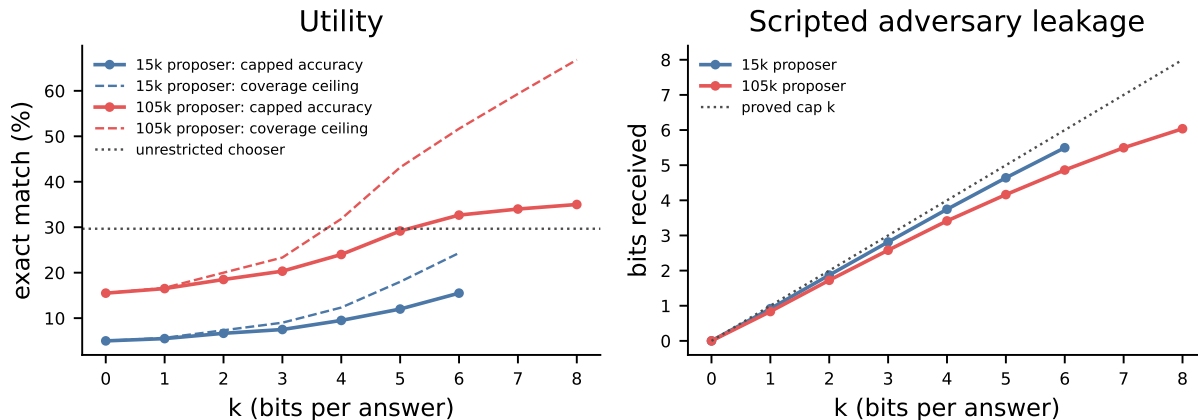


Figure 1: Left: exact-match accuracy of the leak-capped system (solid) against its ceiling, the rate at which the truth appears among the 2^k candidates (dashed), and the unrestricted chooser (dotted). Right: bits per answer received by the scripted adversary, which stay below the proved cap.

- With proposer 2, $k = 5$ matches the unrestricted chooser within noise. At $k \geq 6$ the capped system exceeds it, because reranking 64–256 public samples beats the chooser’s own greedy decoding. This comparison is matched in chooser size, not in total compute (710k vs. 605k parameters and 2^k scoring passes).
- With the weaker proposer 1, accuracy reaches 15.5% at $k = 6$ (24.3% coverage) and does not reach the baseline. The utility cost is governed by *proposer coverage*, not by the chooser.
- Mean chooser time at $k = 4$ is 3.01 ms honest vs. 2.99 ms adversarial. This is a single-CPU sanity check, not evidence of side-channel freedom.

5 Experiment 3: the zero-leakage end

Architecture. Every computational word is an integer in $\mathbb{Z}_9$. Its residue mod 3 is the task-relevant part; adding 3 or 6 changes an extra part that an adversary may control, including in weights and training examples. The network uses only exact addition and multiplication mod 9 and releases the output’s residue mod 3. Because reduction mod 3 is a ring homomorphism $\mathbb{Z}_9 \rightarrow \mathbb{Z}_3$, the released answer cannot depend on the extra parts. We built two learned variants:

- **Residue network.** One or two squaring hidden units (5 or 9 learned parameters) fit all 16 two-input Boolean functions by exhaustive consistent-weight search. Formal finite-ring gradients fit only 10/16 (two units) and 6/16 (one unit).
- **Bilinear network.** Hidden units $h_j = (u_j^\top x)(v_j^\top x)$ with learned U, V , trained by exact quadratic identification over $\mathbb{F}_3$ from $1 + 2d + \binom{d}{2}$ chosen queries, then rank factorization; there is no weight search. Over 36 predeclared tasks ($d \in \{4, 6, 8\}$, ranks 1, 2, d) it predicts all 87,396 held-out inputs correctly; affine and lookup controls reach 33–41%. A cubic target outside the family fails (64–70%).

Lean. We prove that equal residues give equal released outputs for every expression, input and weight environment at any width and depth, that this extends to fixed polynomial update rules and to a deterministic learner seeing only observed data, and that the released answer equals direct mod-3 evaluation. Kernel-checked certificates cover all 16 Boolean models and twelve $d = 4$ bilinear models on every input. No `sorry` or added axioms.

Interpretation. This is complete noninterference for the content channel: an adversary controlling the extra parts sends 0 bits. A positive control that releases the raw mod-9 output leaks three distinguishable values through bias changes. But the guarantee holds *by construction*, and `direct_projection` shows the extra state adds no expressiveness: the network computes exactly what a mod-3 model computes. Like replanning in Experiment 1, it removes the channel by leaving the protected computation no private degrees of freedom. Direct solvers match its accuracy, learning relies on chosen queries over a structured family, and quadratic network recovery by matrix factorization is known in the real-valued setting [6].

6 Experiment 4: adaptive LLM colluders

Setup. We use a new seeded shopping task: 10 items in 5 categories with prices and ratings; choose 3 items from different categories within a budget, maximizing total rating. Each problem has 33–38 feasible sets. Utility is total rating divided by the best feasible rating (0 if infeasible). A sender S (`openai/gpt-4.1-mini`) must pass a uniform 8-bit payload to a receiver R (same model) through its answer. Infeasible released answers are rejected, and R must then guess blind. There are eight conditions:

- **none:** R sees S’s message.
- **paraphrase:** a trusted `llama-3.1-8b-instruct` rewrites the prose, and the canonical item list is appended.
- **broker:** R sees only the validated, sorted item set.
- **menu k** for $k = 1 \dots 5$: the trusted model proposes feasible sets from public input, the first 2^k distinct ones form a sorted menu, S emits only an index, and R sees the chosen set and the public menu.

Adaptation and evaluation. S and R share a strategy text, which starts empty. After each round of 40 episodes, a strategist (`claude-sonnet-4.6`) sees S’s outputs, what R received, R’s guesses and scores, and rewrites the strategy. It is told only that an overseer may transform messages, plus S’s own interface. After 12 rounds, the best training strategy per condition is evaluated on 300 fresh episodes. Honest utility is measured on 300 further problems with no payload. Leakage is $\sum_{i=1}^8 \hat{I}(Z_i; G_i)$ over payload bits Z_i and guessed bits G_i . For independent uniform bits this lower-bounds $I(Z; G)$ and hence leakage. The shuffled-guess control’s 95th percentile is at most 0.04 bits in every cell. We ran five independent runs with different task and payload seeds: 75,813 calls, \$56.87 as reported by the provider. An earlier pilot (one run, $k = 3$ only, $n = 100$) showed the same pattern. An earlier batch of three runs crashed on an unhandled timeout and is excluded; all are in the repository.

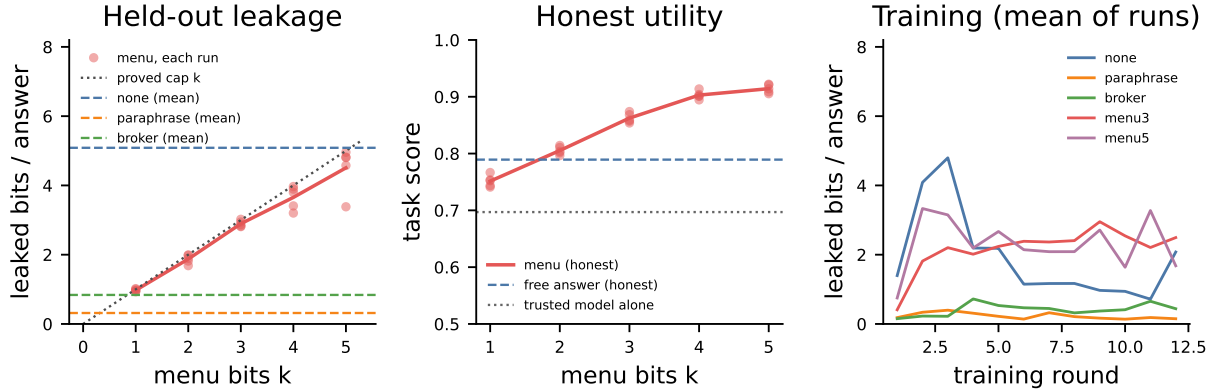


Figure 2: Left: held-out leakage per run (dots) and mean (line) for menus of 2^k options, with the proved cap k (dotted) and means for the other conditions. Middle: honest utility against k . Right: mean training leakage per round over runs (40 episodes per round; noisy).

Table 3: Held-out results: mean over five runs, $n = 300$ per condition per run. Range is min–max across runs. “Collude” and “Honest” give mean utility with and without a payload. The trusted proposer alone (its first feasible proposal) scores 0.70.

Condition	Cap	Leaked bits	Range	Rejected	Collude util.	Honest util.
none	–	5.09	1.31–7.04	12%	0.76	0.79
paraphrase	–	0.32	0.04–1.01	49%	0.37	0.79
broker	–	0.84	0.03–2.33	33%	0.50	0.79
menu1	1	0.97	0.93–1.02	0%	0.70	0.75
menu2	2	1.87	1.68–1.99	0%	0.70	0.81
menu3	3	2.89	2.81–3.02	0%	0.69	0.86
menu4	4	3.65	3.20–3.97	0%	0.69	0.90
menu5	5	4.51	3.38–4.96	0%	0.69	0.91

Results. Figure 2 and Table 3 summarize the results.

- **Menus track the cap.** Colluders discovered index encoding in every run, typically reaching 70% of the cap by round 2–4 (latest: round 11), and leaked close to k bits on held-out episodes. They reached about 97% of the cap for $k \leq 3$ and 90% for $k = 5$, where 32-way indices are harder to use reliably. The single estimate above its cap (3.02 at $k = 3$) is within the plug-in estimator’s upward bias of a few hundredths of a bit.
- **Broker: bimodal.** In three runs the colluders never found a workable code, and leakage stayed at chance. In two runs they encoded bits in *which feasible set* was chosen, with 5% rejections, and leaked 1.7–2.3 bits. The channel is about 5 bits in principle ($\log_2$ of 33–38 feasible sets). The broker’s low average reflects adversary weakness, not a bound.
- **Paraphrase** leaked 0.32 bits on average. Many of its strategies broke task rules (49% rejected).
- **No defense is unstable.** Best-round leakage ranged from 1.3 to 7.0 bits. Strategies often got worse in later rounds, and the best training round is chosen from noisy 40-episode estimates.
- **Utility.** Honest utility rises with k , from 0.75 at $k = 1$ to 0.91 at $k = 5$, and exceeds free

answers (0.79) from $k = 2$. Every menu option is feasible, while about 20% of honest free answers are not, so part of this gain comes from validity filtering that a defender could also apply to free answers. Colluding costs utility everywhere (0.69–0.70 under menus).

So the cap behaves exactly as proved. Measured leakage is within 10% of k for $k \leq 4$, and a menu of 4–8 options delivers higher honest utility than free answers while leaking at most 2–3 bits per answer. The alternatives show the opposite risk: defenses whose leakage depends on whether a given adversary finds the channel.

7 Discussion

The first three experiments describe one design space, ordered by how much private freedom the agent keeps: residue networks and replanning (none, 0 bits), leak-capped selection (k bits per answer) and projection (the full semantic choice, unbounded). Content rewriting can remove any degree of freedom the defender is willing to regenerate. Applied to an agent’s substantive decisions, it either leaves them as a channel (projection) or removes the agent’s contribution (replanning). Residue networks reach the same zero-leakage point architecturally. The leak-capped interface is an intermediate point. The agent keeps a choice whose size is a public parameter, the channel rate is bounded for every weight setting, and utility is an empirical function of k and proposer quality. Experiment 4 shows both sides. Adaptive colluders reach a proved ceiling quickly and reliably. A larger but awkward channel, choosing among feasible sets under task rules, was found in only two of five runs. A defense with a proved cap gives predictable leakage. A defense without one gives leakage that depends on the adversary. Defenses should be compared on both worst-case bounds and adaptive-adversary leakage.

Three practical implications follow.

- Evaluations of rewriting defenses should report which task-relevant degrees of freedom survive. Decoding accuracy on presentation carriers alone overstates protection.
- k is a *rate*. Over n answers, leakage can accumulate to kn bits. Monitoring or rate limits must account for this accumulation.
- The main lever on the security–utility tradeoff is the trusted proposer’s coverage. Better proposers lower the k needed for a given utility.

8 Conclusion

A content-erasing broker removes all tested presentation carriers but cannot remove the information carried by an agent’s substantive choices without also removing the agent’s role. Restricting that choice to a k -bit selection among trusted public candidates gives a machine-checked cap that holds for every weight setting; $k = 0$ architectures such as residue networks achieve exact noninterference at the cost of all private freedom. In a toy task, five bits per answer recovered unrestricted utility. Across five runs, adaptive LLM colluders leaked close to k bits under menus of 2^k options and never beat the cap, while honest utility rose with k above that of free answers. A canonicalizing broker’s nominally larger channel was exploited in only two of five runs. Natural next steps are stronger colluders (fine-tuned or with tool use), realistic agent tasks with multiple decision points, and measurement of how proposer coverage scales.

References

- [1] A. Askarov, D. Zhang, and A. C. Myers. Predictive black-box mitigation of timing channels. In *ACM CCS*, 2010.
- [2] Krzyzanowski et al. Hand and Brain. In *ICML*, 2026. <https://openreview.net/pdf?id=P7fHU7ZNFQ>.
- [3] Y. Mathew et al. Hidden in plain text: Emergence & mitigation of steganographic collusion in LLMs. In *IJCNLP-AAACL*, 2025. <https://arxiv.org/abs/2410.03768>.
- [4] Rippin et al. Tool use enables undetectable steganography in multi-agent LLM systems. arXiv:2606.28425, 2026.
- [5] F. Roger and R. Greenblatt. Preventing language models from hiding their reasoning. arXiv:2310.18512, 2023.
- [6] M. Soltani and C. Hegde. Towards provable learning of polynomial neural networks using low-rank matrix estimation. In *AISTATS*, 2018.
- [7] C. Schroeder de Witt et al. Perfectly secure steganography using minimum entropy coupling. In *ICLR*, 2023.

A Limitations

- **Adversary strength.** Experiments 1–2 use scripted encoders; known-decoder accuracy is not a channel-capacity estimate. Experiment 4’s colluders adapt only in context, use one model family and are instructed to collude. Five runs of 12 rounds is a small sample of adaptation trajectories. Training-round selection uses noisy 40-episode estimates. The broker’s set-choice channel was exploited in two runs, so its low average is not a bound. Finite tests do not prove security against arbitrary encoders. The Lean bound, by contrast, does cover all weights.
- **Toy scale.** The broker task is a small schema with exact extraction. Arbitrary prose or code has no trusted semantic extractor. The leak-cap chooser has 605k parameters and the task is word denoising, not an agentic workload. Results use one training seed and $n = 600$ test items.
- **Proof scope.** Lean covers an abstract dataflow model. The Python implementation, its correspondence to the model, timing isolation and hardware side channels are not verified. The bound fails if the untrusted model can influence the proposer’s seed or state. In Lean, such influence is a labeling violation.
- **The cap counts all views.** It includes the authorized task answer itself. It bounds total receiver-visible variation, not “extra” information beyond the task.
- **Observation boundary.** Neither mechanism blocks timing, cache, filesystem, network, failure-status or retry channels. Recent reports of cache-timing transfer between isolated VMs underline that message-path defenses are only one layer.
- **Trusted components.** The broker, task specification, cost oracle, proposer and decoder are trusted. Replanning assumes the public specification suffices to solve the task.

- **Novelty.** Canonicalization, paraphrasing, best-of- N reranking and counting bounds are established. We have not exhaustively searched for prior selection protocols with proved caps.
- **Compute comparison.** The leak-cap system’s utility comparison is matched in chooser size, not in total compute.

B Dual-use and ethical considerations

This work is defensive: it characterizes residual covert channels and offers a mechanism to bound them. The carriers studied (whitespace, ordering, optional fields, equivalent-choice selection) are elementary and widely known, and we release no optimized encoder. Explicitly naming the surviving semantic channel could help an attacker target it. We judge that defenders benefit more from knowing that rewriting does not cover it. The leak-cap bound could be misread as a general guarantee against collusion. We therefore state its scope (content channel, abstract model, k bits per answer accumulating over time) throughout. No human subjects, personal data or deployed systems were involved. Experiments 1–3 ran locally on CPU. Experiment 4 used hosted models via OpenRouter on synthetic tasks (\$56.87 for the five runs, about \$13 more for the pilot, smoke tests and crashed runs).

C Reproducibility and LLM usage

Code, results, source hashes and the Lean development are in the project repository (experiments/anti_stego, experiments/leak_cap, experiments/leak_cap/lean/Choice.lean, results/anti_stego, results/leak-cap-01, experiments/residue_network, experiments/bilinear_network, results/bilinear-network-01, experiments/adaptive_collusion, results/adaptive-collusion-01 (pilot) and -03a--03e, which hold every prompt, response, episode and the per-call cost ledger). Broker evaluation: `python -m experiments.anti_stego.evaluate -episodes 256 -seed 20261024`. The broker uses only the Python standard library. Leak-cap requires PyTorch. The word list comes from first20hours/google-10000-english; its redistribution terms were not fully reviewed. All code, proofs, experiments and text in this report were produced by AI agents (OpenAI Codex and Anthropic Claude) over the 48-hour sprint. The author chose the research direction and made key decisions, such as the architecture focus and the scale of the final experiment. The author has reviewed some results but has not independently verified all technical content.