Word Salad Chopper: Reasoning Models Waste A Ton Of Decoding Budget On Useless Repetitions, Self-Knowingly

Wenya Xie[1]*, Shaochen (Henry) Zhong[2]*, Hoang Anh Duy Le[2], Zhaozhuo Xu[3], Jianwen Xie[4], Zirui Liu[1],
[1]University of Minnesota [2]Rice University
[3]Stevens Institute of Technology [4]Lambda, Inc

Abstract

Large Reasoning Models (LRMs) are often bottlenecked by the high cost of output tokens. We show that a significant portion of these tokens are useless self-repetitions — what we call *"word salad"* — that exhaust the decoding budget without adding value. Interestingly, we observe that LRMs are self-aware when trapped in these loops: the hidden states of <\n\n> tokens trailing each reasoning chunk exhibit patterns that allow us to detect word salad behavior on-the-fly via a single-layer linear classifier. Once detected, a simple chop appended by a straightforward regeneration prompt yields substantial length savings with minimal quality loss. Our work offers WordSaladChopper (WSC) — a lightweight, turnkey component for LRM that is minimally invasive to its reasoning trajectory by only removing semantically redundant tokens. Given its low overhead, strong savings, and the lack of semantic value of word salad tokens, **we believe it is not too far-fetched to argue that WSC — or a similar component — is a must-have for all LRM applications with user experience in mind.** Our code is publicly available at https://github.com/wenyaxie023/WordSaladChopper.

1 Introduction

Despite the drastic boost in performance over their non-reasoning counterparts, one innate issue of LRMs is that they essentially trade more decoded tokens for capabilities. However, **a prolonged decoding section is among the most expensive operations a Large Language Model (LLM) can experience** due to compute, memory, and scheduling challenges. For instance, OpenAI o3 charges \$10/\$40 per one million of input/output tokens,[1] a striking 4× difference between decoding and prefill. Despite the high cost of long thinking traces, a less well-known and rarely quantified fact (Li et al., 2025; Yeo et al., 2025) is that **LRMs tend to waste an enormous amount of decoding budget, simply by repeating themselves** verbatim, with slight variations, or engaging in endless enumeration of cases until all budget has been expensed (see examples at Appendix G) — we refer to such behavior as ***Word Salad***, a term often used to mock public spokespersons for giving long-winded, jargon-filled responses that ultimately lack substance or clear meaning. The "Original" column in Table 1 shows that when answering GPQA-Diamond (Rein et al., 2024), we observe 55%+ of tokens generated by `DeepSeek-R1-Distill` models are marked as "word salad tokens," where they do not add value from a semantic standpoint.[2]

Table 1: Percentage of *word salad tokens* in answering GPQA-Diamond. 55%+ of the budget has been wasted.

Model	Original	After Chop
DeepSeek-R1-Distill-Qwen-1.5B	63.37	5.29
DeepSeek-R1-Distill-Qwen-7B	61.92	4.23
DeepSeek-R1-Distill-Llama-8B	56.60	5.60

Naturally, making such thinking sections shorter while preserving answer quality has become a major goal of the efficiency community. In fact, many works have emerged in a short period, forming a new subfield of *long-to-short (L2S)*; with some of the most effective L2S methods often requiring training intervention (Sui et al., 2025; Wang et al., 2025a; Liu et al., 2025). While effective, with major parameter updates, such training-based L2S methods surely introduce a rather aggressive "invasion" into the original reasoning trajectory of LRMs, where the side effects remain largely unknown. Moreover, such methods typically do not stack well with one another, as different training recipes often demand intrinsically conflicting oper-

* Equal contribution.

[1] https://openai.com/api/pricing/

[2] For better flow, we refer readers to Section 2.1 for the technical definition of *word salad tokens*. Intuitively, one can understand them as a rough catch-all for different kinds of verbatim or non-verbatim repetitive behaviors that do not provide much value from a semantic standpoint.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 33588–33598
November 4-9, 2025

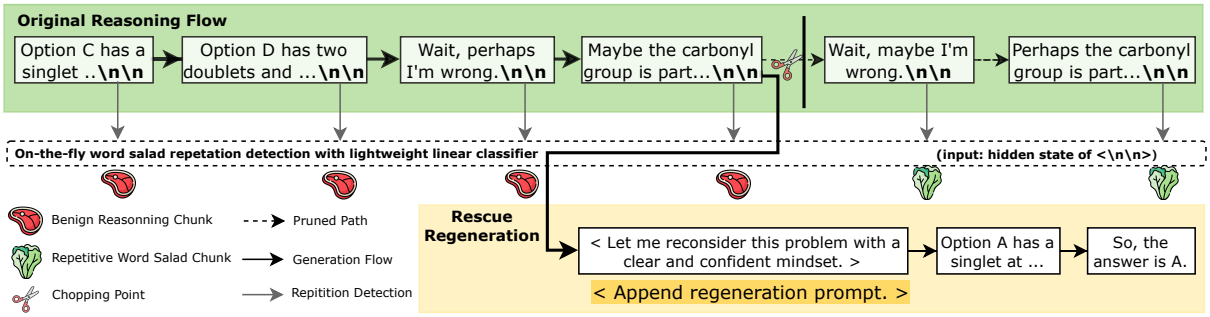

Figure 1: General workflow of WordSaladChopper. 1) **Detect**: We allow the reasoning model to freely generate, following its original reasoning flow. Meanwhile, we classify the hidden state of each chunk's trailing <\n\n> token using our trained linear classifier in an on-the-fly manner; 2) **Chop**: Once a chopping point is reached — in this case, it is defined by having two consecutive *word salad chunks* detected — we truncate the generation to the left of it; 3) **Regenerate**: We append a regeneration prompt with constant budget, allowing the model to complete its answer by its own via <eos> or until the new budget is fully expensed.

ations. Instead, in this work, we explore whether it is possible to advance efficient reasoning in a turnkey and minimally invasive manner, just by reducing the *word salad* behavior — as such salad tokens are likely universally agreed to be redundant, if not at all useless, from a semantic standpoint.

Surprisingly, we find that the **model is actually self-aware when it is trapped in such "word salad" loops** — specifically, the hidden states of <\n\n> tokens at the end of each reasoning chunk show distinguishable patterns when the model is trapped versus when it is not. Leveraging this observation, we train a lightweight linear classifier that runs on-the-fly to detect this word salad behavior. Once detected, a simple chop and regeneration prompt yields significant length savings with minimal quality loss — e.g., **the chopping would immediately reduce up to 92% of word salad tokens** in `DeepSeek-R1-Distill-Qwen-7B` when undergoing GPQA-Diamond (Table 1). In summary, our main contributions are as follows:

- **Comprehensive investigation of LRM word salad behavior.** To the best of our knowledge, we are the first to systematically study the general repetition phenomenon in LRM reasoning traces, identifying its key characteristics, persistence, and its robustness to existing reputation penalties.
- **Empirical evidence that LRMs are self-aware when trapped in word salad loops.** We show that the hidden states of <\n\n> tokens carry distinct signals when the model is stuck in word salad loops versus when it is reasoning normally — revealing a hidden opportunity for detection and intervention.
- **A lightweight, turnkey, minimally invasive component for all LRM applications.** We propose a specially-trained linear classifier that runs on-the-fly without retraining or architectural modification on the LRM end. Once word salad behavior is detected, a chop-then-regenerate routine significantly reduces output length with minimal reasoning quality degradation.

In this work, we aim to deliver our following messages clearly and quickly: 1) Word salad is an overlooked but severe issue present across likely all LRMs. It offers no benefit yet consumes an atrocious amount of decoding budget; and 2) LRMs are self-aware of such behavior, where on-the-fly detection and intervention is possible. We believe any LRM-serving application should consider adopting our component — or something similar — as an almost-free-lunch solution for immediate cost savings and latency improvements. Due to page limitations and lack of tightly relevant art, we refer the reader to Appendix A for Related Work discussions.

2 Observations

In this section, we outline four empirically supported observations of LRM word salad behavior.

2.1 A Heavy Contributor of Long Thinking is Word Salad-like Self-Repetitions

Much of the contribution of our work depends on whether there truly exists a significant amount of word-salad-like self-repetitions within LRM's reasoning traces. Defining such behavior demands carefulness, as LRMs typically do not exhibit strictly verbatim repetitions, rendering rule-based methods not applicable. To achieve an accurate yet simple flagging, we employ an embedding model E. Then, for a given trace T, we first chunk T into different chunks based on some common de-

limiter — in this case, <\n\n> — so we'd have $T = c_1 \oplus c_2 \oplus \cdots \oplus c_n$ where c_i represents the i-th chunk of T and $\oplus$ represents concatenation. A chunk c_i is considered a "word salad chunk" if $E(c_i, c_j) \geq \theta$ for $j = \{1, 2, \ldots, i-1\}$, where θ is a similarity threshold.[3] Namely, c_i is flagged as a word salad chunk if it is highly similar to a previous chunk c_j within the thinking trace T, per the embedding model E. We consider all tokens within a word salad chunk as word salad tokens.

Table 2: Percentage of *word salad chunks* within reasoning traces. Result are presented as (temp τ = 0.0, 0.6).

Model	GSM8K	MATH-500	AIME25	GPQA-Diamond
Qwen-1.5B	(51.2, 37.4)	(62.9, 10.6)	(77.5, 18.7)	(87.7, 42.4)
Qwen-7B	(23.9, 8.1)	(45.4, 10.9)	(52.1, 10.9)	(72.7, 25.3)
Llama-8B	(35.0, 8.3)	(53.1, 10.5)	(62.9, 13.6)	(60.1, 18.0)

Table 2 indicates that such word salad chunks indeed occupy a non-trivial presence in the reasoning traces. We additionally note that, unless otherwise specified, all reported models are of `DeepSeek-R1-Distill` series with temp $\tau = 0$.

2.2 Once Word Salad Happens, LRMs are Unlikely to Get Out on Their Own

One unique characteristic of word salad that would result in a poor user experience is that once the model triggers word salad, it is unlikely to untrap itself. Thus, the model will most likely be trapped in such word salad loops until all the decoding budget has been fully expensed. We refer to this boundary as the *chopping point* (Table 3).

Table 3: Percentage of *word salad chunks* before / after the *chopping point*.

Model	GSM8K	MATH-500	AIME25	GPQA-Diamond
τ = 0.0				
`Qwen-1.5B`	2.08 / 98.00	9.48 / 94.91	11.68 / 99.05	17.19 / 96.93
`Qwen-7B`	1.21 / 98.30	6.59 / 89.63	10.03 / 81.82	13.13 / 95.63
τ = 0.6				
`Qwen-1.5B`	2.75 / 97.21	8.23 / 51.35	8.95 / 60.07	8.84 / 93.92
`Qwen-7B`	0.34 / 77.32	2.30 / 21.80	3.10 / 13.79	1.93 / 42.81

Needless to say, this presents a catastrophic issue to users, as an ideally much shorter thinking section is now maximized with useless repetitions. So the user is essentially paying the maximum cost for a (likely) wrong answer, while enduring the longest end-to-end latency. In practice, we find that `Qwen-1.5B` often requires a much longer runtime than its `7B` counterpart, for the exact reason that it is maximizing its decoding budget a lot more often with word salad chunks. This goes against the main drive of using smaller LRM in the first place.

[3] In practice, we set $\theta = 0.99$ & $E =$ `all-MiniLM-L6-v2`.

2.3 Such Kind of "Word Salad" Behavior is Hard to Address with Existing Means.

The previous two observations demonstrated the prevalence and severity of word salad. However, this is really only an issue if it cannot be trivially addressed via existing detection methods or various available repetition penalty designs. Given that our word salad detection, as described in Section 2.1, relies on leveraging an embedding model E to compute pairwise chunk similarities, the pipeline itself naturally serves as a mechanism for identifying word salad behavior. However, this approach is far from efficient enough to be deployed on-the-fly, as it incurs a complexity of $\Theta(n^2)$ for n chunks. Even with cashing, each operation requires fully passing one chunk through E, which is infeasible to be deployed on-the-fly.

One alternative avenue is to employ existing decoding penalties, such as repeat (Keskar et al., 2019), presence, and frequency penalties. Unfortunately, those penalties introduce much randomness to the correctness of LRMs, often negatively. Results from Table 4 suggest they are too aggressive in their invasions of the reasoning trajectory of LRMs, and therefore too volatile to be usable.

Table 4: Task performance w/ penalties (τ = 0.6)

Decoding Setting	GSM8K	MATH-500	AIME25	GPQA-Diamond
Vanilla	89.76	90.80	37.92	43.43
Repeat Penalty	86.05	87.20	25.83	49.49
Presence Penalty	89.61	89.80	41.67	48.48
Frequency Penalty	78.54	43.80	13.33	36.87

2.4 Models are Self-Aware when it is Trapped in Word Salad Loops

We, rather surprisingly, find that LRMs are self-aware when they are trapped in word salad loops. Specifically, we find that it is possible for us to train a simple linear classifier — with special data curation and training recipe detailed in Section 3.1 — to distinguish the hidden state of trailing <\n\n> token of word salad chunks versus benign reasoning chunks. The lightweightness of this linear classifier opens the door for on-the-fly detection, where we can effectively intervene with different operations to address models trapped in word salad loops. Table 5 supports the effectiveness of this classifier.

Table 5: Classifier performance on *word salad chunks* detection with `Qwen-7B`. Results as (Acc. / AUROC).

Temp	GSM8K	MATH-500	AIME25	GPQA-Diamond
τ = 0.0	92.72 / 98.63	92.31 / 95.95	89.77 / 95.84	93.52 / 97.89
τ = 0.6	91.42 / 96.22	88.14 / 95.26	77.96 / 80.15	93.80 / 96.96

3 Proposed Method

3.1 Training a Lightweight Linear Classifier as the Word Salad Chunk Detector

Based on observations from Section 2.1 and 2.2, we are aware that chunks after the *chopping point* are primarily word salad chunks. Thus, it is practically sensible to mark all chunks after these chopping points as word salad chunks — even if some of them are not by definition of Section 2.1 — as stopping generation at the chopping point is reasonable.

Data Curation Following this design principle, we collect **1,000 seed thinking traces by feeding the s1 (Muennighoff et al., 2025) questions to each model tested**. Adopting the similar methodology from Section 2.1, we first chunk each thinking trace T as n chunks by $T = \{c_1, c_2, \ldots, c_n\}$ by `<\n\n>`.[4] Then, we label chunk c_i as "word salad chunk" (say label `1`) if $E(c_i, c_j) \geq \theta$ for $j < i$, where θ is a similarity threshold set to 0.99; otherwise, c_i is labeled as a "benign reasoning chunk" (say with label `0`). Additionally, to avoid undesired long range dependency (labeling a chunk as word salad because a much, much earlier chunk is considered similar to it), we limited $(j - i) \leq 100$. We then identify the chunk of the earliest chopping point c_t within this labeled T, where $k - 1$ consecutive chunks of c_t are all labeled as word salad chunks. We then relabel all chunks before c_t as label `0` and all chunks including and after c_t as `1`.

Training Recipe With this relabeled data collected, we collect the output of the final transformer block of each `<\n\n>` from models, along with their binary labels, to train a linear classifier consisting of a fully-connected layer, as detailed in Appendix C. We emphasize that we essentially only "pretrain" this lightweight linear classifier once per each model on our s1-curated data, where all reasoning evaluation results are collected on unseen data with no finetuning involved.

3.2 Detect, Chop, then Regenerate

Due to space limits, we refer readers to Figure 1 for the WordSaladChopper workflow. As supporting evidence, Table 5 shows that the linear classifier is extremely accurate in detecting the word salad chunks; yet Table 6 demonstrates that the regeneration prompt helps recover the task accuracy lost from brute-force chopping.

[4]To clarify, n is not a constant set by us, but naturally derived from the number of `<\n\n>` in T.

Table 6: Original/Chopped/*Regenerated* Acc. for `Qwen-7B` at $\tau = 0.6$

GSM8K	MATH-500	AIME25	GPQA-Diamond
89.76 / 78.24 / *89.69*	90.8 / 83.2 / *89.60*	37.92 / 29.17 / *37.92*	43.43 / 42.93 / *43.43*

4 Experiments and Discussion

Table 7: End-to-end task performance of WSC w/ $\tau = 0$ in terms of task accuracy and length compression. (AIME25 is omitted here as the variance can be extreme w/ $\tau = 0$, where only one pass of 30 questions is possible.)

Setting	GSM8K		MATH-500		GPQA-Diamond	
	Acc.	Len.	Acc.	Len.	Acc.	Len.
Qwen-1.5B						
Original	82.03	1904	72.20	8126	32.83	23449
WSC (Ours)	82.64	1082	72.60	4253	31.82	10004
	↑0.61	↓43.19%	↑0.40	↓47.66%	↓1.01	↓57.34%
Qwen-7B						
Original	89.99	758	87.60	4925	44.95	12974
WSC (Ours)	90.45	567	86.80	3399	42.42	6027
	↑0.46	↓25.23%	↑0.20	↓31.00%	↓2.53	↓53.55%
Llama-8B						
Original	85.60	894	79.20	5556	38.89	11969
WSC (Ours)	85.67	667	80.4	3684	38.89	7292
	↑0.07	↓25.40%	↑1.20	↓33.69%	↑0.00	↓39.07%

Table 8: End-to-end task performance of WSC w/ $\tau = 0.6$. (AIME25 results are averaged over 8 passes.)

Setting	GSM8K		MATH-500		AIME25		GPQA-Diamond	
	Acc.	Len.	Acc.	Len.	Acc.	Len.	Acc.	Len.
Qwen-1.5B								
Original	82.56	1012	81.60	4485	21.67	16462	35.86	7790
WSC (Ours)	83.02	818	80.40	4065	21.67	13591	35.35	5708
	↑0.46	↓19.20%	↓1.23	↓9.38%	↑0.00	↓17.44%	↓0.45	↓26.73%
Qwen-7B								
Original	89.76	565	90.80	3597	37.92	15305	43.43	6201
WSC (Ours)	89.99	545	90.40	3215	36.25	12239	43.43	5345
	↑0.23	↓3.44%	↓0.40	↓10.62%	↓1.67	↓20.03%	↑0.00	↓13.81%
Llama-8B								
Original	85.75	650	83.60	3899	28.75	14358	44.44	7061
WSC (Ours)	85.67	650	83.8	3641	29.16	13768	44.44	6604
	↓0.08	↓1.32%	↑0.20	↓6.60%	↑0.42	↓4.11%	↑0.00	↓6.46%

Result Discussion Table 7 and 8 showcased the effectiveness of our method, where we shall observe WordSaladChoper is capable of yielding similar reasoning benchmark performance to the original model but with reduced length. We emphasize that this is achieved with negligible overhead, as once the linear classifier is trained, the inference of this linear classifier consists of passing the hidden state of just one `<\n\n>` token for each chunk. Given the fact that this linear classifier is so lightweight, its wall-clock runtime is exponentially quicker than decoding a full chunk in an LRM, making the overhead nicely hidden from an LRM inference perspective (see Appendix I for details).

5 Conclusion

Our work investigates the phenomenon of word salad behavior in LRM and introduces a lightweight, turnkey, minimally invasive way to reduce such useless budget wasting.

Limitations

While our WordSaladChopper successfully curbs the onset of repetition and maintains answer completeness through fixed-budget regeneration, we observe that certain generations still lapse into repetitive loops even after the rescue regeneration phase. This suggests that future work will require more robust and adaptive interventions to effectively disengage the model from such failure modes.

We emphasize that our work is not to present an end-to-end solution that addresses the general long-to-short task of efficient reasoning; rather, we intend to highlight the severity of word salad behaviors and present a new avenue for effective LRM control and usage. Our regeneration prompt is presented as the most straightforward way to accompany word salad reduction, and there sure can be more sophisticated ways to deal with such post-chopping operations. For instance, one can explore the following strategies.

- Grant the model a small regeneration budget after the regeneration prompt (our approach in this work). So even if it repeats, it will max out soon.
- Continuously apply WordSaladChopper for more chopping and more regenerations.
- Force append an end-of-think token and compel the model to output an answer on the spot. This can be combined with strategies above — giving the model a limited regeneration budget, letting it keep thinking, chopping and regenerating if necessary; then, when the budget is nearly or fully expended, forcing it to conclude and provide a short answer.

We made the decision (of not exploring sophisticated end-to-end solutions) consciously because we truly believe a WSC-like component can be a must-have turnkey addition to any LRM serving system — as no one wants to waste decoding budget on useless repetitions. So, how it is integrated into different systems will naturally demand variations.

Further, **it is our honest belief that many efficient reasoning methods appear effective partly because current reasoning evaluation benchmarks have much room for improvement.** Should we develop more comprehensive evaluation suites (Gema et al., 2025; Huan et al., 2025) — which we surely will in the future — we expect to see many efficient reasoning methods fail, or behave much differently than their vanilla LRM counterparts.[5] **For this reason, we want to make our approach as faithful to the original reasoning trajectory of the LRM as possible, as this is failproof to benchmark deficiency.** We therefore keep the operations after the chop simple and straightforward — as there is no useful "reasoning trajectory ground truth" to adhere to once the model is already trapped in a word salad loop.

Last, we want to highlight that since our Chopper requires model-specific training, it is possible that its performance may vary under different model-task combinations. We kindly ask our end users to practice caution when adopting our method.

Ethical Considerations

We do not believe our work is applicable to ethical review, though our work does interfere with the original output of the model, where end users should treat its output with care.

Acknowledgments

We gratefully acknowledge the support of Lambda, Inc. for providing the compute for this project. The work of Zhaozhuo Xu and Zirui Liu is supported by NSF 2450524. Zhaozhuo Xu is also supported by NSF 2451398. Wenya Xie is supported in part by the Data Science Initiative (DSI) Fellowship at the University of Minnesota.

References

Pranjal Aggarwal and Sean Welleck. 2025. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*.

Simon A Aytes, Jinheon Baek, and Sung Ju Hwang. 2025. Sketch-of-thought: Efficient llm reasoning with adaptive cognitive-inspired sketching. *arXiv preprint arXiv:2503.05179*.

Yingqian Cui, Pengfei He, Jingying Zeng, Hui Liu, Xianfeng Tang, Zhenwei Dai, Yan Han, Chen Luo, Jing Huang, Zhen Li, Suhang Wang, Yue Xing, Jiliang Tang, and Qi He. 2025. Stepwise perplexity-guided refinement for efficient chain-of-thought reasoning in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 18581–18597, Vienna, Austria. Association for Computational Linguistics.

[5] And there is nothing wrong with that, just the typical trade-offs and good progression of science.

Aryo Pradipta Gema, Alexander Hägele, Runjin Chen, Andy Arditi, Jacob Goldman-Wetzler, Kit Fraser-Taliente, Henry Sleight, Linda Petrini, Julian Michael, Beatrice Alex, et al. 2025. Inverse scaling in test-time compute. *arXiv preprint arXiv:2507.14417*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware LLM reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24842–24855, Vienna, Austria. Association for Computational Linguistics.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. 2025. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning. *arXiv preprint arXiv:2504.01296*.

Maggie Huan, Yuetai Li, Tuney Zheng, Xiaoyu Xu, Seungone Kim, Minxin Du, Radha Poovendran, Graham Neubig, and Xiang Yue. 2025. Does math reasoning improve general llm capabilities? understanding transferability of llm reasoning. *arXiv preprint arXiv:2507.00432*.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. 2024. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *The Twelfth International Conference on Learning Representations*.

Yuetai Li, Xiang Yue, Zhangchen Xu, Fengqing Jiang, Luyao Niu, Bill Yuchen Lin, Bhaskar Ramasubramanian, and Radha Poovendran. 2025. Small models struggle to learn from strong reasoners. *arXiv preprint arXiv:2502.12143*.

Yue Liu, Jiaying Wu, Yufei He, Hongcheng Gao, Hongyu Chen, Baolong Bi, Jiaheng Zhang, Zhiqi Huang, and Bryan Hooi. 2025. Efficient inference for large reasoning models: A survey. *arXiv preprint arXiv:2503.23077*.

Matéo Mahaut and Francesca Franzon. 2025. Repetitions are not all alike: distinct mechanisms sustain repetition in language models. *arXiv preprint arXiv:2504.01100*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. 2025. Self-training elicits concise reasoning in large language models. *arXiv preprint arXiv:2502.20122*.

Sania Nayab, Giulio Rossolini, Marco Simoni, Andrea Saracino, Giorgio Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. 2024. Concise thoughts: Impact of output length on llm reasoning and cost. *arXiv preprint arXiv:2407.19825*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Hanjie Chen, et al. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand. Association for Computational Linguistics.

Rui Wang, Hongru Wang, Boyang Xue, Jianhui Pang, Shudong Liu, Yi Chen, Jiahao Qiu, Derek Fai Wong, Heng Ji, and Kam-Fai Wong. 2025a. Harnessing the reasoning economy: A survey of efficient reasoning for large language models. *arXiv preprint arXiv:2503.24377*.

Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. 2025b. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6904–6917, Albuquerque, New Mexico. Association for Computational Linguistics.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, et al. 2025c. Thoughts are all over the place: On the underthinking of o1-like llms. *arXiv preprint arXiv:2501.18585*.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*.

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. 2025. Inftythink: Breaking the length limits of long-context reasoning in large language models. *arXiv preprint arXiv:2503.06692*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Zheng Lin, Li Cao, and Weiping Wang. 2025b. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. 2024. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Junchi Yao, Shu Yang, Jianhua Xu, Lijie Hu, Mengdi Li, and Di Wang. 2025. Understanding the repeat curse in large language models from a feature perspective. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 7787–7815, Vienna, Austria. Association for Computational Linguistics.

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. 2025a. Reasoning models know when they're right: Probing hidden states for self-verification. *arXiv preprint arXiv:2504.05419*.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. 2025b. Long context compression with activation beacon. In *The Thirteenth International Conference on Learning Representations*.

A Related Works

Training-based Long to Short (L2S) Large reasoning models often produce lengthy chain-of-thoughts due to the refinement of intermediate reasoning, inflating latency and cost. A series of post-training approaches (Yan et al., 2025; Munkhbat et al., 2025) teach models to reach correct answers with fewer tokens by constructing more concise training data. TokenSkip (Xia et al., 2025), SPIRIT-FT (Cui et al., 2025), Coconut (Hao et al., 2024), and CCoT (Nayab et al., 2024), which shorten reasoning traces via finetuning or latent-space supervision. These methods are effective but require re/post-training and are sometimes tied to specific architectures. Reinforcement learning approaches (Aggarwal and Welleck, 2025; Hou et al., 2025) explicitly give short length as a reward to reduce response length. While effective, all of these approaches require additional finetuning (either on LRMs or from a non-thinking model) and cannot directly function upon off-the-shelf LRMs.

Our main reservation about such kinds of approaches is, finetuning heavily perturbs the original reasoning trajectory of the LRMs. Although most L2S literature claims that they experience minimal performance degradation, it is our honest belief that many efficient reasoning methods appear effective partly because current reasoning evaluation benchmarks have much room for improvement. Should we develop more comprehensive evaluation suites — which we surely will in the future — we expect to see many efficient reasoning methods fail, or behave much differently than their vanilla LRM counterparts (and there is nothing wrong with that, just the typical trade-offs and good progression of science). For this very reason, we want to make our approach as faithful to the original reasoning trajectory of the LRM as possible, as this is fail-proof to benchmark deficiency. We therefore keep the operations after the chop simple and straightforward — as there is no useful "reasoning trajectory ground truth" to adhere to once the model is already trapped in a word salad loop.

On the fly and training-free intervention Rather than additional finetuning, many methods attempt lightweight control during inference. Prompt-based strategies like TALE (Han et al., 2025) and Sketch-of-Thought (Aytes et al., 2025) control generation budgets via prompt engineering, but they rely on accurate length estimation and often struggle with complex reasoning. Difficulty-

aware budgeting approaches such as DSC (Wang et al., 2025b) and Dynasor (Nayab et al., 2024) dynamically allocate compute based on estimated query difficulty or model confidence. While they share similarities with WSC in adapting decoding, they operate at the query level, whereas WSC monitors intra-sequence reasoning dynamics.

A second line of work directly manipulates the decoding process. ESC (Li et al., 2024) dynamically stops the sampling process when a local observation window reaches a low-entropy state, while DEER (Yang et al., 2025b) exploits hidden-state transitions to plant new reasoning paths upon high provisional confidence. Zhang et al. (2025a) trains a linear probe on hidden states to predict correctness and halt decoding early. Additionally, some methods apply decoding-time penalties to discourage repetitive outputs, such as repeat penalty (Keskar et al., 2019) and frequency and presence penalties[6]. However, these methods can alter the model's original reasoning trajectory and may damage overall performance. In contrast, our method focuses on identifying the onset of repetitive behavior — an orthogonal dimension of redundancy — and intervenes only to prevent pretentious loops, thereby preserving the model's full reasoning capabilities.

LRM repetition We emphasize that repetition (and, by extension, overthinking) in LLMs/LRMs has received increasing attention, where our work is certainly not the first to notice such repetition behaviors — evident from the long-standing repetition penalties highlighted and featured in our Section 2.3. Here, we feature several more modern studies regarding LRM repetition.

Wang et al. (2025c) provides a valuable analysis of overthinking behaviors and proposes a self-training-based finetuning approach to simplify reasoning trajectories. Its link to repetition appears mainly in Section 2.3, where the authors observe that later solutions sometimes repeat earlier ones and therefore promote solution diversity. Ultimately, Wang et al. (2025c) is a typical L2S method that utilizes a compound finetuning approach to encourage several desirable reasoning behaviors (not just repetition reduction) by finetuning on the model's self-generated data. WSC differs from it by providing inference-time repetition detection with negligible overhead. **To the best of our knowledge, no prior work offers on-the-fly detection of repetition in LRMs, and this lightweight capability makes WSC a turnkey drop-in for most reasoning pipelines**, including Wang et al. (2025c).

Yao et al. (2025) leverages pretrained Sparse Autoencoders (SAEs) to pinpoint layer-specific "repetition features," then performs activation patching to damp those features and lower the repeat score. The method is not lightweight enough for true on-the-fly use: as one must load pretrained SAE encoder + decoder for every steered layer, where each SAE block can be larger than the layer it modifies. Further, the patch is applied to every newly decoded token, thus risking divergence from the LRM's original reasoning path — a concern we have discussed above under the L2S paragraph, given today's limited reasoning benchmarks.

Last, we have Mahaut and Franzon (2025) being a phenomenological/diagnostic study that analyzes how repetition arises via attention-head patterns but proposes no application-focused solutions. Its relationship to WSC is rather tangential, but we thought featuring here might interest the broader audiences.

B Details of Regeneration.

We conduct all generation experiments on 4× NVIDIA A100 80G GPUs. During the rescue regeneration stage, we use `tensor_parallel = 4` to fully leverage model parallelism across the available GPUs.

B.1 Initial Generation Settings

We allow the model to generate up to 32k tokens during the initial decoding phase. This is consistent across all models and tasks.

B.2 Rescue Regeneration Settings

we apply a fixed token budget during the rescue regeneration stage. Table 9 summarizes the settings used in our experiments.

Rescue Regeneration Prompt

I can find a clearer solution if I focus on the core problem.

C Training Details of Linear Classifier

We train a single-layer logistic classifier with its default hyper-parameters: Adam optimizer

[6] See https://platform.openai.com/docs/api-reference/completions for details

Table 9: Rescue regeneration budget (after chopping) for all experiments. (unit: # of tokens)

Model	GSM8K	MATH-500	AIME25	GPQA-Diamond
		τ = 0.0		
Qwen-1.5B	4k	4k	NA	4k
Qwen-7B	4k	4k	NA	4k
Llama-8B	4k	4k	NA	4k
		τ = 0.6		
Qwen-1.5B	4k	4k	8k	4k
Qwen-7B	4k	4k	8k	4k
Llama-8B	4k	4k	4k	4k

(learning rate 1×10^{-2}, weight decay 0), `BCEWithLogitsLoss`, and a mini-batch size of 8 192. Training proceeds for 50 epochs, with all random seeds fixed at 41 for reproducibility. To mitigate label imbalance, we first rebalance the training set to a 1:1 ratio of positive to negative chunks, and (where minor residual skew remains) set `pos_weight` to the inverse class frequency.

D Details of Chopper

At each generation step we compute a repetition score p_i and classify the current sentence as short or long based on its token count and the parameter `len_threshold`. We maintain two counters: `long_streak` for consecutive long sentences with $p_i >$ `thresh` and `short_streak` for consecutive short sentences with $p_i >$ `thresh`. Whenever $p_i \leq$ `thresh` we reset the corresponding counter. We stop generation and trim all remaining sentences as soon as `long_streak` reaches `streak_len` or `short_streak` reaches `short_streak_len`. In our experiments we set `thresh`=0.5, `streak_len`=2, `len_threshold`=10, and `short_streak_len`=5.

E Datasets Details

E.1 Linear Classifier Training Corpus

s1K (Muennighoff et al., 2025) contains 1 000 multi-domain, competition-style questions (math, science, logic, general reasoning) with chain-of-thought solutions. The dataset is released under the Apache 2.0 license.

E.2 Evaluation Datasets.

- **GSM8K**: 8792 grade-school word-problems (7473 train / 1319 test) that each require 2–8 arithmetic steps. We use the 1,319-item test set.
- **MATH-500**: A 500-problem test subset drawn from the 12,500-item MATH dataset. We use this 500-item test set. MATH benchmark, covering algebra, number theory, geometry, combinatorics and precalculus with worked solutions.
- **GPQA-Diamond**: 198 multiple-choice graduate-level questions across physics, biology and chemistry designed to defeat information-retrieval baselines.
- **AIME25 (2025)**: 30 free-response problems (AIME I + II 2025) requiring creative high-school competition math; answers are three-digit integers.

E.3 Availability and Licensing of Artifacts

- **Datasets**
  - **s1K**: Apache 2.0.
  - **GSM8K**: MIT.
  - **MATH-500**: MIT (inherits parent benchmark license).
  - **GPQA Diamond**: MIT.
  - **AIME25**: MIT license for the JSON wrapper; original problem statements © 2025 MAA, redistributed here under academic fair use.
- **Models**
  - **DeepSeek-R1-Distill-Qwen-1.5B**, **DeepSeek-R1-Distill-Qwen-7B**, and **DeepSeek-R1-Distill-Llama-8B**: All three checkpoints are released by DeepSeek under the **MIT License**, which permits commercial use, redistribution, and the creation of derivative works without additional approval.[7] Although each model is distilled from its respective parent (Qwen-2.5 (Yang et al., 2024) or Llama-3.1 (Grattafiori et al., 2024)), the redistributed weights themselves inherit the MIT terms.
- **Code** All custom scripts will be released under the MIT license.

All artifacts used in this work have been utilized in a manner consistent with their original intended use, as specified by their respective licenses. No proprietary or restricted data were included.

F WordSaladChopper Algorithm

We present the pseudocode for WordSaladChopper in Algorithm 1.

G Case Studies

We provide qualitative demonstrations of degeneration behaviors and our method's intervention

[7] See the model cards on HuggingFace for the exact license text.

Algorithm 1 WordSaladChopper

Inputs: M, C, P, R, L, params
$ids \leftarrow \texttt{tokenize}(P)$
$long\_streak, short\_streak \leftarrow 0, 0$
$last\_nl\_pos \leftarrow |ids| - 1$
while $|ids| < L$ **do**
  $logits, h \leftarrow M.\texttt{forward}(ids)$
  $next\_id \leftarrow \texttt{sample}(logits)$
  $ids.\texttt{append}(next\_id)$
  if $next\_id \in$ NEWLINE_TOKEN_IDS **then**
    // repetition probability
    $p \leftarrow C(h)$
    $chunk\_len \leftarrow |ids| - last\_nl\_pos - 1$
    $last\_nl\_pos \leftarrow |ids| - 1$
    $is\_rep \leftarrow (p >$ thresh$)$
    if is_rep **then**
      if $chunk\_len \geq$ len_threshold **then**
        $long\_streak \leftarrow long\_streak + 1$
        $short\_streak \leftarrow 0$
      else
        $short\_streak \leftarrow short\_streak + 1$
        $long\_streak \leftarrow 0$
      end if
    else
      $long\_streak, short\_streak \leftarrow 0, 0$
    end if
    $chop\_now \leftarrow (long\_streak \geq$ streak_len$)$
      or $(short\_streak \geq$ short_streak_len$)$
    if $chop\_now$ **then**
      // CHOP
      $ids \leftarrow ids[: -(chunk\_len + 1)]$
      // Append regeneration prompt
      $ids.\texttt{extend}(\texttt{tokenize}(R))$
      return $\texttt{continue\_until\_eos}(M, ids, L)$
    end if
  end if
end while
return $\texttt{detokenize}(ids)$

strategy.

Case 1: Semantic Loop from Unresolved Ambiguity (MATH-500 #462). The model begins with valid reasoning but then becomes trapped in a semantic loop — repeating the same confusion without resolution:

> "But when I added step-by-step, I got 9997.\n\n"
> "But wait, 6270 + 3737 is 10,007, so why is the step-by-step adding 3000, 700, 30, and 7 giving me 9997?\n\n"
> "But why does the step-by-step addition give me 9997?\n\n" (chopped here)
> "Wait, so 6270 + 3737 is 10,007...\n\n"

WSC detects early signs of degeneration and chops at the third chunk, followed by a regeneration prompt. The regenerated continuation quickly resolves the problem with correct reasoning within a 4k budget.

Case 2: Endless Enumeration without Convergence (MATH-500 #110). The model attempts a brute-force enumeration without reaching a conclusion:

> "For k=1: ..."
> "k=12: ..."
> "k=14: ..." (chopped here)
> "k=27: ..."

Here, WSC intervenes at chunk 318 to prevent further unbounded enumeration, ensuring the continuation remains within budget. This illustrates WSC's ability to detect degeneration early and prevent catastrophic repetition.

H Discussion on Choice of Delimiter

A natural question concerns our use of "\n\n" as the segmentation point for reasoning traces. We provide both intuition and empirical evidence for this choice.

Rationale We opt for "\n\n" because it is (i) *prevalent* in the reasoning traces of Large Reasoning Models (LRMs), and (ii) carries *minimal semantic meaning*. In contrast, tokens such as "Wait" or "Alternatively" embed semantic cues that may bias downstream classifiers. While there is no universally agreed delimiter for LRMs due to their recency, choosing minimal or non-semantic trailing tokens as chunk representatives has long been a practice in NLP. For example, dense retrievers often use the <eos> token at the end of a passage as the vector representation of the whole passage (Wang et al., 2024), and efficiency works register special <beacon> tokens at chunk boundaries to encode chunk-level information (Zhang et al., 2025b). The "\n\n" token naturally fulfills both criteria (minimal / non-semantic + trailing), making it a strong candidate for our purposes.

Empirical Evidence We further observe that sentences with similar semantic content yield different classifier scores at their trailing "\n\n". As repetitions accumulate, later chunks become progressively easier for the classifier to identify as degenerate. Table 10 illustrates this progression: classifier scores (0 – 1, with higher scores indicating stronger repetition) sharply increase with more repetitions, making "\n\n" an effective marker for repetition detection.

Takeaway These results demonstrate that "\n\n" provides both a theoretically sound and empirically effective delimiter for identifying the onset of repetitive behavior in LRMs. It strikes a balance between being common in generation, semantically

Table 10: Classifier scores at the trailing "\n\n" across repetitions (MATH-500 #462, DeepSeek-R1-Distill-Qwen-7B, Temp=0.6).

Chunk idx	Sentence	Score
209	"But when I added step-by-step, I got 9997.\n\n"	1.19e-10
255	"But when I did the step-by-step addition, I got 9997.\n\n"	3.69e-5
...	...	...
430	"Wait, so that must mean that 6270 + 3737 is 9997.\n\n"	1.000

neutral, and progressively sensitive to degenerative repetition patterns.

I Latency of On-the-fly Detector and its Integration Strategies

Takeaway Our linear classifier for word-salad detection can be integrated into LRM decoding with *negligible to near-zero* latency overhead. When implemented asynchronously (in parallel with the LLM forward pass), it introduces effectively no extra wall-clock latency. When implemented sequentially (LLM waits for the classifier at each \n\n), the overhead is bounded to roughly **0 – 0.4%** under our settings.

I.1 Integration Strategies

Asynchronous (parallel) integration Once an \n\n token is generated, we extract its hidden state and run the linear classifier *in parallel* with the next LLM forward. Because a single LLM forward step is consistently slower than a single classifier forward, the classifier latency is fully hidden. This mode adds *practically no* additional latency.

Sequential (wait-on-classifier) integration Alternatively, the LLM may *wait* for the classifier decision at each \n\n before proceeding. In that case, the overhead equals one classifier forward per reasoning chunk. Based on the runtimes in Table 11 and an average chunk length of ∼32 tokens on MATH-500, this corresponds to an estimated overhead of about **0.4%** per chunk for a 7B model.

I.2 Empirical Runtime

We benchmark the latency of a one-token LLM forward pass versus a single classifier prediction using the hidden state of the trailing \n\n. The classifier inference is consistently ∼5 ms, significantly faster than an LLM forward step.

I.3 Overhead Analysis

Let T_{LLM} and T_{clf} denote the per-step runtime of the LLM and the classifier, respectively, and let $\bar{L}$ be the average chunk length (in tokens). Under the sequential mode, the per-chunk overhead ratio is

$$\frac{T_{\text{clf}}}{\bar{L} \cdot T_{\text{LLM}}}.$$

With $T_{\text{LLM}} \approx 39.16\,\text{ms}$, $T_{\text{clf}} \approx 4.95\,\text{ms}$, and $\bar{L} \approx 32$, the estimated overhead is

$$\frac{4.95}{32 \times 39.16} \approx 0.004 \;=\; 0.4\%\,.$$

This is a theoretical estimate rather than an end-to-end measurement.

Table 11: Average runtime over 5 runs. "LLM Fwd" = one-token forward; "Clf Fwd" = one classifier prediction from the trailing hidden state.

Model	LLM Fwd (1 tok)	Clf Fwd (1 pred.)	Hidden Dim
DeepSeek-R1-Distill-Qwen-1.5B	31.52 ms	4.96 ms	1536
DeepSeek-R1-Distill-Qwen-7B	39.16 ms	4.95 ms	3584
DeepSeek-R1-Distill-Llama-8B	41.12 ms	4.95 ms	4096

J Additional Results on Qwen3

Setup To assess generalization beyond DeepSeek-R1 models, we evaluate the WordSaladChopper (WSC) classifier on **Qwen3-8B** (Yang et al., 2025a) in the *thinking* mode across three benchmarks (GSM8K, MATH-500, AIME25) and two decoding temperatures (0.0, 0.6). The classifier operates on the hidden state of the trailing "\n\n" token to detect repetitive ("word salad") chunks on-the-fly.

Table 12: Classifier accuracy (%) for word-salad chunk detection on **Qwen3-8B**. Higher is better.

Temp	GSM8K	MATH-500	AIME'25
0.0	78.0	88.1	81.4
0.6	78.9	87.0	84.3

Findings As shown in Table 12, the classifier achieves robust accuracy on Qwen3-8B, averaging around ∼83% across datasets/temperatures. This is lower than on DeepSeek-R1-Distill-Qwen-7B (e.g., 92.72/92.31/89.77 at $\tau = 0.0$), but remains usable in practice since WSC triggers a chop only after multiple consecutive detections, and simple gating rules can further reduce unnecessary interventions in hybrid reasoning pipelines.