Score × Decoder: A Unified View of Unsupervised Inference-Time Scaling for Hallucination Mitigation

Anonymous ACL submission

Abstract

Large language models hallucinate even when the answer lies within their parameters. While inference-time scaling can surface this latent knowledge, the most effective methods require supervision: a trained verifier or reward model. We ask what can be done with only a base language model: which intrinsic signal best identifies correct outputs, and how should it be decoded? We cast this as a score × decoder grid pairing four scores (perplexity, contrastive, power-distribution likelihood, and self-verification) with three decoding families (optimization, sampling, consensus), and evaluate every cell on MATH500 with the base and instruction-tuned Qwen3-1.7B. While self-verification, which prompts the model to judge its own answer and is sharpened by a training-free virtual-thinking prefix, works well in most settings, no score has a fixed quality: its value depends on the decoder that consumes it and on model capability. When no supervision is available, the score and the decoding family must be chosen together.

1 Introduction

Large language models hallucinate. However, to mitigate these errors, we must first distinguish between two fundamentally different pathologies: the answers a model *could* have gotten right but didn't, and the ones it *couldn't* have gotten right but won't admit. Within the taxonomy of the know–unknow quadrant (Yin et al., 2023; Li et al., 2025), these manifest as *unknown knows*, elicitation failures where the correct knowledge is trapped in the model's weights but fails to surface, and *unknown unknowns*, awareness failures where the model lacks the knowledge entirely yet blindly fabricates. While traditionally treated as separate problems evaluated by distinct metrics (pass@1 for elicitation versus calibration, abstention, or selective accuracy for awareness), both failure modes share a highly compelling virtue: they can, in principle, be intercepted and corrected at inference time without the prohibitive cost of retraining. Elicitation failures can be rescued by better navigating the generation space through strategic sampling, whereas awareness failures can be neutralized by robustly estimating the model's internal confidence.

We focus in this work on the *unknown knows* subset: elicitation failures, where the correct answer is already latent in the model's weights. The inference-time literature has tools for this (verifier-guided search, learned reward models, best-of-N with trained scorers), but the most effective ones share a dependency: a *supervised* signal, typically a verifier or reward model trained on labeled correctness data. This conflates whether inference-time compute can help with whether a good verifier happens to be available, and restricts deployment to domains where such labels are abundant. We ask the strictly unsupervised version: what can be recovered using only scalar signals extracted from the base LM itself, with no learned verifier, no reward model, no external judge?

The central design choice in this setting is which scalar signal extracted from the base model best identifies correct outputs, and how that signal should be composed into a decoding strategy. We evaluate four *gray-box* scoring functions as parallel candidates: perplexity, contrastive scores (Li et al., 2023; O'Brien and Lewis, 2024), power distribution likelihood, and self-verification (Kadavath et al., 2022; Xie et al., 2023; Weng et al., 2023) with a virtual thinking prefix, and compose each with three decoding families: *optimization*, which searches for the highest-scoring candidate (realized as best-of-N); *sampling*, which draws from a score-shaped target distribution $p_{\text{final}}(y \mid x) \propto p_{\text{LM}}(y \mid x)\, s(y \mid x)$ (realized as SMC and MCMC); and *consensus* ($a^\star \in \arg\max_a \mathbb{E}_{y\sim p_{\text{final}}}[\mathbf{1}[\text{ans}(y) = a]]$), which selects the mode of p_{final} estimated over many samples and is realized as majority vote (Wang et al., 2023) and score-weighted aggrega-

tion. This factorization unifies a wide range of prior inference-time methods into a single score $\times$ decoder grid.

We make three contributions.

(1) Score design. We identify self-verification with a *virtual-thinking* prefix as a structurally better-grounded unsupervised elicitation score than likelihood-derived alternatives. It is *conditioned on the full candidate* and *semantically aligned* with the downstream metric (asking whether an answer is right, not whether it is likely), and is defined as the normalized probability of the correct verdict, $s_{\mathrm{sv}}(y \mid x) = p_{\checkmark}/(p_{\checkmark}+p_{\times})$ (see Section 3), making it invariant to the absolute scale of verdict token probabilities. Self-verification is not new as a signal in isolation (Kadavath et al., 2022; Xie et al., 2023); our contribution is to make it the central score in a unified inference-time framework and to enhance it with virtual thinking.

(2) Virtual thinking without finetuning. We show that the calibration of self-verification can be sharpened by inserting a sequence of tokens before the verdict logits that consumes additional compute without producing visible reasoning. We find that the choice of prefix matters: *structured* fillers such as countdowns ("$50, 49, 48, \ldots, 1$") outperform *uniform* fillers such as dots ("$\ldots$"). This departs from prior findings that latent-reasoning filler tokens require training to be useful (Pfau et al., 2024; Goyal et al., 2024); we argue the gap is structural: verifier-side virtual thinking is easier than generator-side virtual thinking because the verifier only needs to produce a single classification bit, not a coherent token sequence.

(3) Systematic evaluation. We evaluate every cell of the score $\times$ decoder grid along three complementary axes that correspond to distinct uses of an inference-time score. *Score quality:* for a fixed pool of candidates, how well does the score rank correct candidates above incorrect ones (local ranking, per-question AUROC and top-k accuracy)? *Decoding effectiveness:* when combined with a score, do optimization methods (best-of-N) or sampling methods (SMC, MCMC) produce more accurate single outputs than vanilla decoding (pass@k, accuracy of the modal sample)? *Aggregation effectiveness:* when compute permits many samples per question, does score-weighted consensus improve over majority voting or best-of-N selection? Across these axes, self-verification with virtual thinking performs well on most evaluations, but no score wins universally. The score $\times$ decoder factorization lets us characterize scores and decoders separately, and find that a score's realized quality is not fixed: it depends on the decoder that consumes it, so when no supervision is available the score and the decoding family should be chosen together.

Scope. Our claims concern the interaction structure between scores and decoders, not absolute rankings across models or domains. Section 7 and Appendix D give mechanistic explanations (dynamic-range mismatch, particle collapse) that should transfer beyond our setup, and we leave empirical confirmation at scale to future work.

2 Related Work

Score Functions for Inference-Time Guidance. Inference-time scoring signals differ in the model access they require; we focus on *gray-box* signals, which read the model's own token-level logits rather than only its text. One line of work builds training-free, unsupervised scores: sequence-level perplexity is the canonical choice, while *contrastive decoding* (Li et al., 2023; O'Brien and Lewis, 2024) contrasts a strong expert's logits against a weaker amateur, recently extended by Kim et al. (2026) to contrast against a perturbed thinking trace. Unsupervised *self-verification* instead prompts the model to judge its own output and reads the verdict logit, e.g. $P(\text{True})$ (Kadavath et al., 2022; Xie et al., 2023). A second, *supervised* line trains external estimators: step-level *process reward models* (PRMs) (Lightman et al., 2024; Wang et al., 2024) provide learned guidance but require expensive annotation and specialized training. This raises the question we study: can an unsupervised score read from the base model alone guide decoding without the annotation cost of a trained verifier? We vary the scoring function across four gray-box candidates (perplexity, contrastive scoring, the power distribution, and self-verification) and map how each interacts with MCMC, consensus reweighting, and SMC guidance.

Sampling-Based Methods. Framing generation as probabilistic inference has motivated a line of sampling-based approaches. Lew et al. (2023) introduce LLaMPPL, a Sequential Monte Carlo (SMC) framework for controlled generation, later extended with learned twist functions that estimate

expected future reward (Zhao et al., 2024) and to syntactic/semantic constraints (Loula et al., 2025). Puri et al. (2025) apply particle-based Monte Carlo to inference-time scaling, reporting 4–16× better scaling than deterministic search but relying on a process reward model. On the MCMC side, Faria et al. (2024) propose a Metropolis–Hastings sampler driven by expected reward for translation. A related line of work biases decoding toward a target distribution without resampling at the sequence level, e.g., gradient-based controlled decoding in discrete token space (Pynadath and Zhang, 2025). Most closely related, the concurrent work of Karan and Du (2026) and Ji et al. (2026) explores MCMC over a sharpened *power distribution* of the base model. Both papers fix this single scoring choice and report strong gains *in the MCMC setting alone*.

Optimization-Based Methods. Search-based generation methods cast inference as an optimization problem over a structured token space. The baseline approach, *best-of-N*, samples multiple candidates and selects the highest-scoring sequence under a verifier, with recent refinements focusing on early-decoding signals to minimize sampling costs (Wang et al., 2026). More complex frameworks like *Tree-of-Thoughts* (Yao et al., 2023) and *RAP* (Hao et al., 2023) extend this concept by employing Monte Carlo Tree Search (MCTS) as a policy-improvement operator over tree-structured reasoning paths. This test-time compute scaling paradigm underpins modern frontier models such as OpenAI's o1 (OpenAI, 2024) and DeepSeek-R1 (Guo et al., 2025).

Consensus-Based Methods. Rather than selecting a single candidate sequence, consensus methods aggregate across multiple decoded samples. Standard approaches like *Self-consistency* (Wang et al., 2023) rely on a simple majority vote among extractable answers, while weighted variants scale votes by confidence signals (Kang et al., 2025) or intermediate path features (Wan et al., 2025). We incorporate consensus aggregation into our study not to propose a novel voting mechanism, but to serve as a diagnostic lens for evaluating scoring functions. By applying a standard post-hoc reweighting using calibrated probabilities derived from each score, we complete the picture of inference-time scaling behavior.

Latent and "Virtual" Thinking. While the above methods differ in how they aggregate or select outputs, they share a reliance on explicit token-level reasoning as thinking; a separate line of work asks whether such thinking can instead be internalized into latent computation. A parallel line of work attempts to internalize the computational benefits of a chain-of-thought into hidden states or special tokens rather than explicit text. Architectural interventions such as pause-tokens (Goyal et al., 2024), meta-tokens (Zelikman et al., 2024), and continuous-thought decoding (Coconut; Hao et al., 2025) all operate in continuous latent spaces. However, current literature paints a pessimistic view of these "virtual" thinking paradigms when applied without heavy training; for instance, Pfau et al. (2024) demonstrate that LLMs fail to benefit from virtual filler tokens unless they undergo dense, highly specific supervision, leading to a consensus that virtual thinking does not work out-of-the-box.

3 Score Function Design

A central ingredient of our framework is a scalar score $s(y \mid x) \in \mathbb{R}^+$ that measures the correctness of a response y to a query x. Rather than relying on an external verifier or a separately trained reward model, we design a score that is *intrinsic*: it is computed by prompting the same language model that generated y to evaluate its own output. Prior work on virtual reasoning has focused on the generator side, where the model must produce coherent token sequences, a task for which plug-and-play filler tokens have been shown to be ineffective without dense supervision (Goyal et al., 2024; Pfau et al., 2024). As we describe below, we instead repurpose virtual thinking as a *scoring-side device*, a structurally simpler use case that succeeds completely plug-and-play. This section describes the two components of the score: the self-verification prompt and the virtual thinking countdown, and defines the resulting scoring function.

3.1 Self-Verification Score

Given a query x and a candidate response y, we construct a verification prompt $v(x, y)$ that instructs the language model to judge whether y correctly solves x. The prompt is designed so that the model's first generated token is one of two special verdict tokens, ✓ (correct) or ✗ (incorrect),[1] followed optionally by a brief explanation; see Appendix C for the full template and grading rules. By

[1]The actual tokens used are the Unicode characters ✓ (U+2705) and ✗ (U+274C); LaTeX renders them here via `\ding{51}` and `\ding{55}` as visual approximations.

terminating the prompt with the incomplete string `"### Verdict: "`, we arrange for the model's next token to be either ✓ or ✗, whose log-probabilities are directly readable from the output distribution. Let $p_{✓}(x, y)$ and $p_{✗}(x, y)$ denote the probabilities that the language model assigns to the verdict tokens ✓ and ✗, respectively, as the next token following $v(x, y)$. We define the *self-verification score* as the odds ratio of the correct verdict:[2]

$$s_{\mathrm{sv}}(y \mid x) \;=\; \frac{p_{✓}(x, y)}{p_{✗}(x, y)}. \tag{1}$$

3.2 Virtual Thinking Countdown

Prompting a model for an immediate verdict is often unreliable because it precludes prior reasoning. To encourage implicit deliberation without triggering an explicit chain-of-thought, we insert a *virtual thinking prompt* before the `"### Verdict:"` line (see Appendix C.2). We evaluate two variants: a **countdown filler** $(n, n{-}1, \ldots, 1)$ and a **dot filler** (d repeated dots). Both allocate input tokens for implicit reasoning before forcing a verdict. Because this filler is part of the input context, s_{sv} maintains its single-forward-pass evaluation cost regardless of length. We compare and ablate these variants in Appendix E.

4 Elicitation Framework

Given a scoring function $s(y \mid x) \in \mathbb{R}^{+}$ that evaluates the quality of a response y to a query x , a natural question is how to best spend additional inference-time compute to improve output quality. We organize the answer around two complementary strategies. The first, *optimization* (Section 4.1), uses the score to search for a single high-quality output. The second, *consensus* (Section 4.3), instead aggregates over a collection of candidate outputs drawn from a score-shaped distribution, trading the commitment of picking one best candidate for the stability of collective agreement. Both strategies are applicable to any scoring function, and can in principle be combined.

4.1 Optimization

The most direct use of a score is to treat inference as a search problem: generate candidate responses and return the one with the highest score. The simplest instantiation is best-of-N, which draws N responses $y_1, \ldots, y_N \sim p_{\mathrm{LM}}(\cdot \mid x)$ [3] independently from the language model and returns

$$y^{*} = \underset{i \in [n]}{\arg\max}\; s(y_i \mid x). \tag{2}$$

More sophisticated search algorithms such as beam search and MCTS can exploit the sequential structure of generation to allocate compute more efficiently, and we leave a thorough empirical comparison to future work.

4.2 Sampling from the Score-Shaped Distribution

Rather than optimizing for a single output, we can define a *score-shaped* target distribution

$$p_{\mathrm{final}}(y \mid x) \;\propto\; p_{\mathrm{LM}}(y \mid x) \cdot s(y \mid x), \tag{3}$$

which up-weights responses that are both likely under the language model and assigned high score. The inclusion of p_{LM} in the target (rather than defining $p_{\mathrm{final}} \propto s$ alone) is a principled design choice whose necessity we discuss in Appendix D. Sampling from p_{final} is non-trivial because the normalizing constant is intractable and the support is the space of all token sequences. In practice, this constant is handled implicitly: in MCMC, it cancels in the Metropolis–Hastings acceptance ratio (Eq. 4); in SMC, the resampling step normalizes particle weights at each step, so no explicit computation of the constant is required.

Markov Chain Monte Carlo (MCMC). MCMC methods construct a Markov chain whose stationary distribution is p_{final}. Concretely, we adopt the *prefix-resample* proposal of Karan and Du (2026), in which a new candidate is generated by sampling a random prefix of the current response and then continuing it autoregressively; we omit the blockwise extension described in that work and refer readers there for the full procedure. The resulting Metropolis–Hastings chain accepts a proposed response y' with probability

$$\alpha = \min\left(1,\; \frac{s(y' \mid x)}{s(y \mid x)}\right), \tag{4}$$

ensuring detailed balance with respect to p_{final}, by specialization of the general Metropolis–Hastings

[2] This score is well-defined provided the model assigns negligible probability to tokens other than ✓ and ✗ at the verdict position. We empirically verify that $p_{✓}(x, y) + p_{✗}(x, y) \approx 1$ holds across inputs in Appendix C.3, confirming that the verification prompt reliably elicits the intended verdict format.

[3] Throughout this paper, $p_{\mathrm{LM}}(y \mid x)$ denotes the probability of generating y given input x under the language model, where x is formatted according to the prompt template described in Appendix C.

acceptance rule (see Eq. 12). After a suitable burn-in period, the chain produces approximate samples from the target distribution.

Sequential Monte Carlo (SMC). Following the basic SMC formulation of Doucet et al. (2001), SMC methods maintain a population of N weighted *particles* $\{y_{1:t}^{(i)}, w_t^{(i)}\}_{i=1}^{N}$ that are propagated token-by-token through the generation process. At each step t, the language model extends each particle by one token under the proposal distribution, and the particle weight is updated as

$$w_t^{(i)} \;\propto\; s(y_{1:t}^{(i)} \mid x), \tag{5}$$

where s is evaluated on the partial sequence $y_{1:t}^{(i)}$. Periodically, a resampling step duplicates high-weight particles and discards low-weight ones, concentrating compute on promising prefixes. At termination, each surviving particle is an approximate draw from p_{final}. [4]

Remark: two roles of p_{LM}. p_{LM} is used in two ways here, and the two are configured independently. As the *proposal*, it draws candidate sequences, where we are free to use practical settings (temperature $\tau < 1$, top-k, top-p) for fluent, diverse candidates. As the basis of the *score*, it supplies the probabilities that $s(y \mid x)$ reads; this is always evaluated at $\tau = 1$ with no truncation, reflecting the Conflating the two would distort s and bias the sampler away from the intended p_{final}. Proposal hyperparameters are in Section 5.

4.3 Consensus

Given a set of candidates, consensus methods select the response that is most representative of the ensemble rather than the one with the highest individual score. Formally, we seek

$$a^* = \arg\max_a \; \mathbb{E}_{y\sim p_{\text{final}}}\big[\mathbf{1}[\text{ans}(y) = a]\big], \tag{6}$$

i.e., the mode of p_{final}. This objective can be estimated in two ways depending on how the candidates were obtained.

[4]The formulation requires the score to be applicable to *partial* sequences. Alternatively, if the score is only meaningful for complete sequences, the weight of a partial sequence $y_{1:t}$ can instead be set to the expected score over possible completions, $w_t \;\propto\; \mathbb{E}\big[s(y \mid x) \mid y_{1:t}\big]$, which can be approximated by averaging the scores of several sampled completions of $y_{1:t}$. We leave this formulation to future work and use partial-sequence scoring in our experiments.

Majority vote for MCMC/SMC samples. When candidates $y_1, \ldots, y_n$ are (approximate) draws from p_{final}, as produced by the MCMC or SMC samplers of Section 4.2, the expectation in Eq. (6) can be estimated directly by unweighted majority vote:

$$a^* \approx \arg\max_a \; \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}[\text{ans}(y_i) = a]. \tag{7}$$

No reweighting is needed because the sampling distribution already matches the target.

Importance-weighted vote for standard LM samples. When candidates are drawn from the base model p_{LM} rather than from p_{final}, a direct majority vote is biased toward regions of high p_{LM} that may not be favored by the score. We correct for this mismatch via importance weighting. By a change of measure,

$$\begin{aligned} &\mathbb{E}_{y\sim p_{\text{final}}}\big[\mathbf{1}[\text{ans}(y) = a]\big] && (8)\\ &= \mathbb{E}_{y\sim p_{\text{LM}}}\left[\frac{p_{\text{final}}(y \mid x)}{p_{\text{LM}}(y \mid x)}\,\mathbf{1}[\text{ans}(y) = a]\right] \\ &\propto \mathbb{E}_{y\sim p_{\text{LM}}}\big[s(y \mid x)\,\mathbf{1}[\text{ans}(y) = a]\big], && (9)\end{aligned}$$

where the second line uses Eq. (3) and drops the normalizing constant, which is the same for all a and does not affect the $\arg\max$. This yields the weighted consensus estimator

$$a^* \approx \arg\max_a \; \sum_{i=1}^{n} s(y_i \mid x)\,\mathbf{1}[\text{ans}(y_i) = a], \tag{10}$$

which reduces to standard majority vote when all scores are equal and to best-of-N when candidates are near-unique.

5 Experiments

5.1 Settings

Dataset. We evaluate on **MATH500** (Lightman et al., 2024), a 500-problem subset of the MATH benchmark (Hendrycks et al., 2021). It is the smallest widely-used benchmark that simultaneously offers verifiable correctness, extractable answers, and a sufficient difficulty range: the properties our score × decoder analysis requires. We score exact-match on the final boxed answer after normalizing whitespace and symbolic forms.

Models. We use the **Base** (pretrained) and **Chat** (instruction-tuned) variants of **Qwen3-1.7B** (Team, 2025). Drawing both from one model family fixes architecture, tokenizer, and pretraining, so any Base-vs-Chat difference isolates the effect of instruction tuning. Each variant is its own generator and scorer (no external judge), and all runs use `bfloat16`. We fix model scale to keep the score $\times$ decoder grid well-populated, since partial coverage at larger scales would not support our interaction claims.

Generation budget and token lengths. All sampling uses $\tau = 0.7$, top-$k = 50$, and top-$p = 0.95$. Chat candidates are 2,048-token continuations following a 2,048-token truncated thinking trace; Base candidates are 4,096-token completions. All post-hoc methods (ranking, optimization, best-of-N, reweighted consensus) share a pool of $N = 100$ candidates per question. SMC runs $N = 100$ particles, bootstrapping every 50 tokens. MCMC runs 100 independent Metropolis–Hastings chains of 10 steps each, retaining the final state of each as one sample.

5.2 Score Functions

In all score definitions below, p_{LM} is evaluated at $\tau = 1$ with no top-k or top-p truncation, consistent with the remark in Section 4.2.

Perplexity. $s_{\mathrm{ppl}}(y \mid x) = p_{\mathrm{LM}}(y \mid x)^{1/|y|}$. The geometric mean of token probabilities, inversely related to sequence perplexity.

Power distribution score. $s_\beta(y \mid x) = p_{\mathrm{LM}}(y \mid x)^\beta$. The base-model distribution sharpened to a power $\beta > 1$. We adopt $\beta = 4.0$ from Karan and Du (2026) but retain our standard generation temperature of 0.7 (rather than the recommended $1/\beta = 0.25$) to keep sampling conditions consistent across all scoring methods.

Contrastive score. $s_{\mathrm{CD}}(y \mid x) = p_{\mathrm{EXP}}(y \mid x)/p_{\mathrm{AMA}}(y \mid x)$. The ratio of expert to amateur likelihoods, amplifying tokens uniquely favored by deliberative reasoning (Li et al., 2023). [5] In the SMC condition only, we use the length-normalized variant $s_{\mathrm{CD}}(y \mid x)^{1/|y|}$ to prevent length bias from eliminating already-completed particles.

Self-verification score. (s_{sv}, Eq. (1), Section 3). We report three variants: a no-virtual-thinking baseline, a **dot filler** with $d = 50$, and a **countdown filler** with $n = 50$. The complete ablation over filler type and length is reported in Appendix E.

5.3 Metrics

We evaluate scores along three axes: candidate *ranking*, distribution *shaping* via score-guided sampling, and *consensus* improvement. Throughout, x denotes a question from set $\mathcal{X}$, Normal/SMC/MCMC pool (of x) denotes candidates $\{y_1, \ldots, y_N\}$ drawn from the corresponding sampling methods.

Ranking. We report **Top-k accuracy**: the fraction of questions for which at least one correct candidate appears among the k highest-scored candidates (k=1 recovers best-of-N accuracy; k=N recovers oracle pass@N), and **per-question AUROC**: the probability that a randomly drawn correct candidate outscores a randomly drawn incorrect one, treating $s(\cdot \mid x)$ as a binary classifier within each question. We report the CDF of AUROC across questions containing at least one correct and one incorrect candidate. Both metrics are evaluated on the Normal pool.

Sampling. We report **pass@k**: the probability that at least one of k i.i.d. draws from the corresponding pool is correct, computed via the standard unbiased estimator (Chen et al., 2021) and averaged over $\mathcal{X}$.[6] We also report mean single-sample accuracy and mean number of distinct answers per question, characterizing the diversity of the distribution induced by each decoding method.

Consensus. We report **consensus accuracy**: the fraction of questions whose aggregated answer is correct, under the methods of Section 4.3.

6 Results

6.1 Ranking Ability

Self-verification ranks highest on nearly every metric: it achieves the best per-question AUROC on both models and recovers the most correct answers at k=1 on the hard subset (Figure 1). Power scores competitively on Chat but poorly on Base; its apparent strength on Chat reflects the model's already-

[5] For both Chat and Base model, $p_{\mathrm{AMA}} = p_{\mathrm{LM}}$. For the Chat model, p_{EXP} is p_{LM} with a thinking trace in the prompt. For the Base model, the p_{EXP} is p_{LM} with a structured analysis prefix (generated with the prompt C.4; truncated to 2,048 tokens) in the prompt.

[6] For Normal decoding and MCMC the k samples are independent—each MCMC output is the final state of a separate chain—whereas SMC particles are correlated through shared ancestry via resampling, so pass@k for SMC should be read with this caveat.

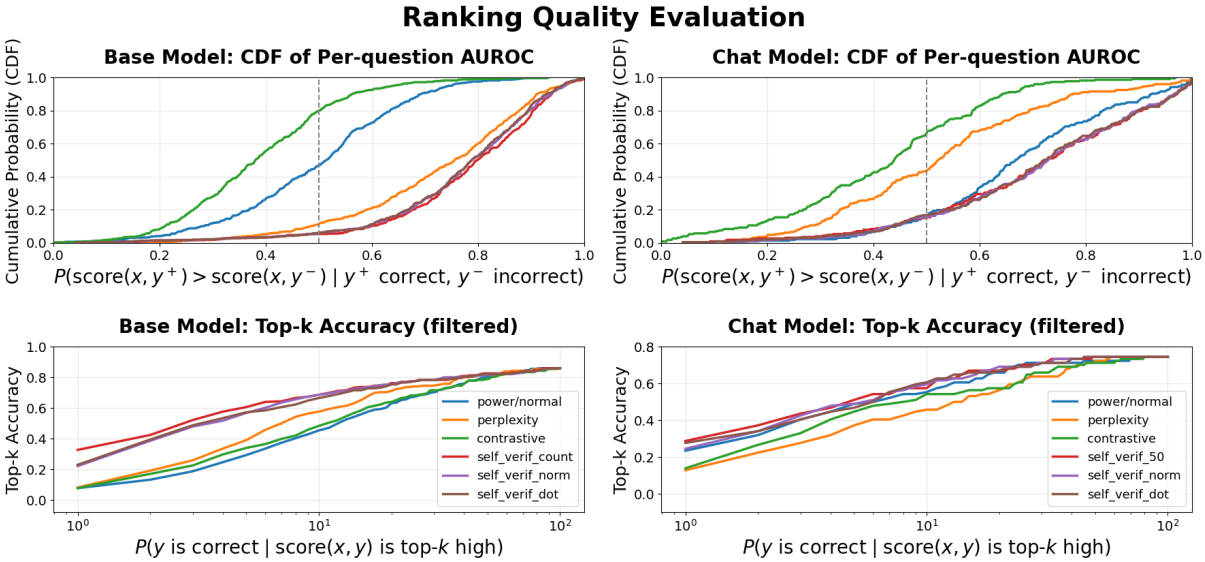

Figure 1: **Ranking quality of different scores** for the Base (left) and Chat (right) models. *Top:* CDF of per-question AUROC over questions with valid pools; mass to the right of the 0.5 line indicates above-chance discrimination (a curve shifted rightward is better). *Bottom:* Top-k accuracy on the hard subset (oracle accuracy $\leq$0.5).

concentrated output distribution rather than genuine discriminative ability. Perplexity offers weak ranking signal on Base and essentially none on Chat. Contrastive is the weakest score throughout; on Base it relies on an engineered analysis prefix as a substitute for the intrinsic expert/amateur pair available on Chat (a thinking-trace vs. bare-prompt continuation), which explains why contrastive underperforms more severely on Base than the other scores.

On the Chat model, power and self-verification are nearly indistinguishable under Top-k accuracy. Figure 1 therefore reports Top-k results on the hard subset (questions with oracle accuracy $\leq$0.5) to expose score differences; the unfiltered version is given in Figure 4, Appendix B.

6.2 Sampling Quality

MCMC variants track or exceed unguided decoding across the full pass@k curve, while SMC variants saturate early (Figure 2). The divergence traces to SMC's resampling: permanently concentrating weight onto a shrinking set of prefixes collapses diversity to 8–30 distinct answers per question (just 1.55 for power-guided SMC), compared to $\approx$100 for MCMC and unguided decoding (Table 1). SMC stakes the outcome on a few high-scoring prefixes while discarding the coverage that makes sampling valuable; when the score is unreliable, no diversity remains to recover. Self-verification paired with MCMC yields the strongest distribution on both models: best Base single-sample accuracy and best Chat pass@k.

	Single-sample Acc.		#Distinct Ans.	
Decoding	**Base**	**Chat**	**Base**	**Chat**
Normal	0.488	0.801	97.63	99.98
PPL(SMC)	0.501	0.818	17.75	26.46
Self-Verif(SMC)	0.337	0.811	13.30	8.20
Contrast(SMC)	0.356	0.805	17.33	30.00
Power(SMC)*	–	**0.822**	–	1.55
PPL(MCMC)	0.498	0.800	99.99	99.99
Self-Verif(MCMC)	**0.552**	0.815	99.99	99.99
Contrast(MCMC)	0.379	0.803	100.0	99.99
Power(MCMC)	0.540	0.809	99.87	99.75

* Excluded for the Base model: produces degenerate non-stopping sequences under this setting.

Table 1: Mean single-sample accuracy and number of distinct answers per question by decoding method.

6.3 Consensus Quality

Self-verification is the most reliable consensus weight: reweighting the Normal pool by s_{sv} yields the highest weighted-consensus accuracy, and it is also the best MCMC consensus guide (Figure 3). The likelihood-derived scores are less stable: the large dynamic range of power and perplexity lets a single high-score candidate dominate the weighted vote and degenerate toward best-of-N, most visibly on the Base model where power-reweighted consensus collapses. SMC-based consensus is competitive only when the SMC pool has not collapsed

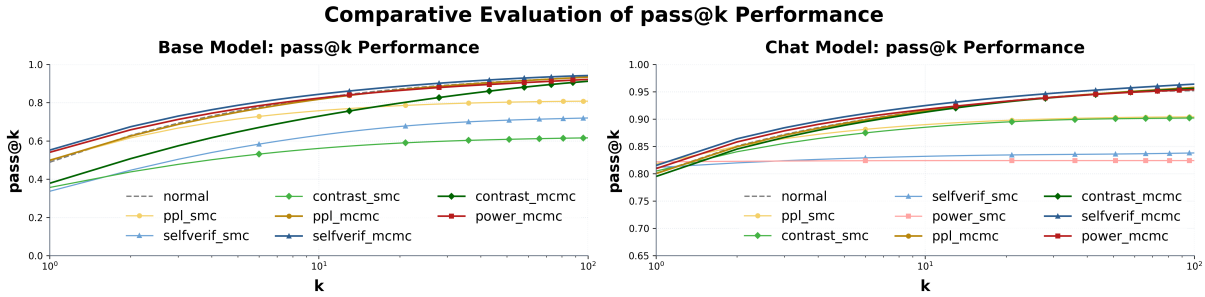

Figure 2: **pass@k by decoding method**, on the Base (left) and Chat (right) models, averaged over $\mathcal{X}$.

onto a few incorrect modes (Table 1). Virtual thinking does not improve consensus: self-verification with and without fillers are near-indistinguishable in consensus accuracy, suggesting its benefit is specific to ranking.

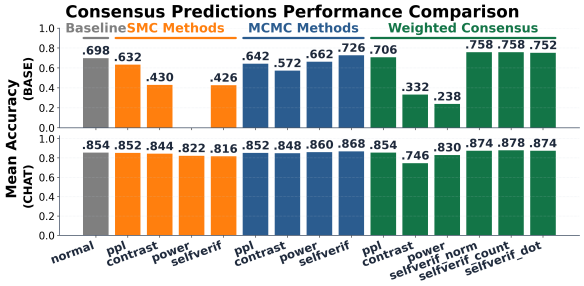

Figure 3: **Consensus accuracy** for Base (top) and Chat (bottom) models, grouped by the aggregation method (baseline is majority vote over Normal pool).

7 Analysis

Decoding choice matters more on Base model. Score-guided methods yield larger gains over unguided decoding on Base than on Chat, plausibly because instruction tuning already concentrates the Chat model's output distribution toward correct answers, leaving less room for any score or decoder to improve upon and compressing observable differences between score functions.

Virtual thinking sharpens top-k only. Virtual thinking moves only Top-k: per-question AUROC and consensus accuracy show no significant difference across variants. At small k, however, the variants separate clearly (an $\approx$9-point gap on Base and $\approx$3-point on Chat at k=1) before converging by $k\approx$15–30. Virtual thinking thus makes the single best candidate more reliably first without improving broader ordering. The optimal filler type is model-dependent: countdown leads at small k on Base, while variants converge on Chat, likely because Chat's reasoning traces already exhibit frequent self-checking behavior, reducing the marginal benefit of explicit countdown cues.

Reweighting a vote is not the same as guiding a sampler. Reweighted consensus aggregates raw score magnitudes, so a single high-scoring outlier can hijack the vote and degrade the weighted vote toward best-of-N behavior. Samplers such as MCMC, by contrast, consume scores probabilistically against competing candidates, preserving diversity rather than surrendering to extremes. This asymmetry explains why a peaky score like power fails under reweighted consensus (where outlier domination goes unchecked) yet succeeds under MCMC, where no single candidate can monopolize selection. Ultimately, a score's value in reweighting is bounded by its best-of-N quality; its value in sampling is not.

8 Conclusion

We studied inference-time scaling for unknown knows, where the correct answer is latent in the model's weights but fails to surface, in the fully unsupervised setting: a base language model with no learned verifier, reward model, or external judge. We evaluated four unsupervised gray-box scores across optimization, sampling, and consensus decoders. Self-verification with a virtual-thinking prefix performs best on most evaluations: without fine-tuning, it ranks candidates reliably, guides MCMC most effectively, and is the most dependable consensus weight. Still, no score has fixed quality: its realized value depends on the decoder that consumes it and on model capability. Concretely, self-verification with MCMC is robust across both models, whereas SMC is fragile and score-weighted consensus needs a stable score. When no supervision is available, the score and decoding family must therefore be chosen together.

Limitations

Scope and transferability. All experiments use Qwen3-1.7B (Base and Chat) on MATH500, and our conclusions should be read within those boundaries. On the model side, scale may shift the picture unevenly: self-verification could improve as larger models tend to be better calibrated, while likelihood-based scores may weaken against a stronger base distribution, so whether score and decoder rankings hold in the 7B–70B regime or under different RLHF recipes remains open. On the domain side, MATH500's verifiable, extractable answers are more favorable to self-verification than open-ended tasks would be, since the verification prompt can straightforwardly ask about format compliance and numerical correctness; whether SV+VT offers comparable advantages on reading comprehension, summarization, or code generation remains to be established. The virtual-thinking findings carry an additional transferability caveat: the relative effectiveness of countdown versus dot fillers is likely sensitive to the model's pretraining distribution and tokenization scheme, and a filler pattern that aligns with frequent subword patterns in Qwen3's vocabulary may not transfer to models with different vocabularies. We leave cross-scale, cross-domain, and cross-tokenizer confirmation to future work.

Compute overhead of self-verification. Self-verification with virtual thinking requires an additional forward pass per candidate to obtain verdict logits, effectively doubling the per-candidate inference cost relative to likelihood-based scores. In our $N{=}100$ pool setting this overhead is absorbed into a fixed compute budget, but it becomes non-trivial in interactive or latency-sensitive settings. We do not report wall-clock runtimes or token-throughput comparisons, and we do not explore whether a cheaper approximation (e.g., a single shared verification pass, or early exit on the filler sequence) can preserve most of the quality gain.

Domain scope of virtual thinking. Our finding that structured countdowns outperform uniform fillers as virtual-thinking prefixes was established on mathematical reasoning benchmarks. It is plausible that the advantage is domain-specific: countdown sequences may prime the model toward the sequential, stepwise computation that arithmetic problems reward, and may confer no analogous benefit (or may even harm calibration) on tasks requiring open-ended retrieval, commonsense reasoning, or long-form generation. We leave the design of domain-appropriate virtual-thinking prefixes to future work.

Optimization-based decoding not yet systematically evaluated. Our framework positions score functions as general-purpose guides for a family of optimization-based decoders (including best-of-N selection, beam search, and tree-search methods such as MCTS), and the scores are defined and motivated in this broader context. However, we do not yet report systematic experiments across these strategies. The ranking metrics in Section 5.3 (top-k accuracy, AUROC) capture a slice of this picture (top-k at $k{=}1$ recovers best-of-N accuracy on a fixed pool), but they do not vary the candidate budget, explore structured search over partial sequences, or compare tree-shaped vs. flat search at matched compute. A complete evaluation would benchmark score-guided beam search and MCTS against unguided sampling across a range of compute budgets and characterize the conditions under which tree-structured optimization outperforms sampling-based alternatives. We leave this to future work.

SMC with fixed resampling schedule. Our SMC implementation uses a fixed, uniform resampling schedule and does not explore variance-reducing alternatives such as learned twist functions or adaptive particle budgets. The SMC results should therefore be interpreted as characterizing naive importance-weighted sampling under each score, not as an upper bound on SMC-based inference.

Fixed MCMC hyperparameters. Our MCMC experiments use a fixed burn-in and chain length of 10 steps across all benchmarks and scores, without ablating sensitivity to these choices. Longer chains or adaptive schedules may yield different results.

Token-level guided decoding not evaluated. Our framework treats scores as sequence-level functionals applied to complete or partial candidates, composed post-hoc with samplers and optimizers. A complementary family of methods intervenes instead at the token level, biasing the next-token distribution at each decoding step, including the original autoregressive formulation of contrastive decoding (Li et al., 2023) and the more recent contrastive thinking decoding (Kim et al., 2026), which contrasts a perturbed thinking trace against a fluent

one. Because these methods modify the generation process itself rather than reranking completed sequences, they sit outside the score $\times$ decoder factorization we study. A direct comparison (in both output quality and compute budget) between token-level guided decoding and our sequence-level framework remains an open question.

Potential risks. While this work targets hallucination *reduction*, one downstream risk merits acknowledgment. The inference-time scaling methods we study substantially increase compute requirements relative to standard decoding, contributing to higher energy consumption; this overhead may also widen the gap between well-resourced and resource-constrained practitioners. We caution against deploying these methods in resource-constrained settings without careful consideration of computational cost.

References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Arnaud Doucet, Nando de Freitas, and Neil Gordon. 2001. *An Introduction to Sequential Monte Carlo Methods*, pages 3–14. Springer New York, New York, NY.

Gonçalo R. A. Faria, Sweta Agrawal, António Farinhas, Ricardo Rei, José G. C. de Souza, and André F.T. Martins. 2024. Quest: Quality-aware Metropolis–Hastings sampling for machine translation. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. 2024. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *Nature*, 645:633–638.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, Singapore. Association for Computational Linguistics.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E Weston, and Yuandong Tian. 2025. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Xiaotong Ji, Rasul Tutunov, Matthieu Zimmer, and Haitham Bou Ammar. 2026. Scalable power sampling: Unlocking efficient, training-free reasoning for LLMs via distribution sharpening. *arXiv preprint arXiv:2601.21590*.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, and 1 others. 2022. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*.

Zhewei Kang, Xuandong Zhao, and Dawn Song. 2025. Scalable best-of-n selection for large language models via self-certainty. In *2nd AI for Math Workshop @ ICML 2025*.

Aayush Karan and Yilun Du. 2026. Reasoning with sampling: Your base model is smarter than you think. In *The Fourteenth International Conference on Learning Representations*.

Taehyeon Kim, Youngsoo Jang, Hyunsoo Lee, Yu Jin Kim, and Moontae Lee. 2026. Contrastive thinking decoding can improve answers for reasoning models.

Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash Mansinghka. 2023. Sequential monte carlo steering of large language models using probabilistic programs. In *ICML 2023 Workshop: Sampling and Optimization in Discrete Space*.

Siheng Li, Cheng Yang, Taiqiang Wu, Chufan Shi, Yuji Zhang, Xinyu Zhu, Zesen Cheng, Deng Cai, Mo Yu, Lemao Liu, Jie Zhou, Yujiu Yang, Ngai Wong, Xixin Wu, and Wai Lam. 2025. A survey on the honesty of large language models. *Transactions on Machine Learning Research*. Survey Certification.

Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. 2023. Contrastive decoding: Open-ended text generation as optimization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12286–12312, Toronto, Canada. Association for Computational Linguistics.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Alexander K. Lew, Tim Vieira, and Timothy J. O'Donnell. 2025. Syntactic and semantic control of large language models via sequential monte carlo. In *The Thirteenth International Conference on Learning Representations*.

Sean O'Brien and Mike Lewis. 2024. Contrastive decoding improves reasoning in large language models.

OpenAI. 2024. OpenAI o1 system card. `https://cdn.openai.com/o1-system-card-20241205.pdf`.

Jacob Pfau, William Merrill, and Samuel R. Bowman. 2024. Let's think dot by dot: Hidden computation in transformer language models. In *First Conference on Language Modeling*.

Isha Puri, Shivchander Sudalairaj, Guangxuan Xu, Abhishek Bhandwaldar, Kai Xu, and Akash Srivastava. 2025. Rollout roulette: A probabilistic inference approach to inference-time scaling of llms using particle-based monte carlo methods. In *Advances in Neural Information Processing Systems*, volume 38, pages 156427–156454. Curran Associates, Inc.

Patrick Pynadath and Ruqi Zhang. 2025. Controlled LLM decoding via discrete auto-regressive biasing. In *The Thirteenth International Conference on Learning Representations*.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. 2025. Reasoning aware self-consistency: Leveraging reasoning paths for efficient LLM sampling. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3613–3635, Albuquerque, New Mexico. Association for Computational Linguistics.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yiming Wang, Pei Zhang, Siyuan Huang, Baosong Yang, Zhuosheng Zhang, Fei Huang, and Rui Wang. 2026. Sampling-efficient test-time scaling: Self-estimating the best-of-n sampling in early decoding. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, Singapore. Association for Computational Linguistics.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. 2023. Self-evaluation guided beam search for reasoning. In *Advances in Neural Information Processing Systems*, volume 36, pages 41618–41650. Curran Associates, Inc.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhangyue Yin, Qiushi Sun, Qipeng Guo, Jiawen Wu, Xipeng Qiu, and Xuanjing Huang. 2023. Do large language models know what they don't know? In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8653–8665, Toronto, Canada. Association for Computational Linguistics.

Eric Zelikman, Georges Raif Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah Goodman. 2024. Quiet-STar: Language models can teach themselves to think before speaking. In *First Conference on Language Modeling*.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Grosse. 2024. Probabilistic inference in language models via twisted sequential Monte Carlo. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

A Ethics Statement

Artifact use. This work uses two existing artifacts: the MATH500 benchmark (Lightman et al., 2024) (a 500-problem research evaluation subset of MATH (Hendrycks et al., 2021)) and the Qwen3-1.7B model family (Team, 2025). Both are used strictly for research evaluation, consistent with their intended purpose. MATH500 is a publicly released research benchmark; our use is limited to model evaluation and we do not redistribute it. Qwen3-1.7B is released under the Apache 2.0 license, which permits research use. We do not use either artifact for any commercial purpose, and any

derivatives produced in this work (e.g., score logs, ranked candidate sets) are intended solely for academic research.

Compute and environmental impact. All experiments use a single small model (Qwen3-1.7B in `bfloat16`) evaluated on a 500-question benchmark, representing a modest computational footprint. We do not train or fine-tune any models.

Potential harms. Our work studies scoring functions for ranking model-generated mathematical solutions. We do not collect human data, handle personally identifiable information, or deploy systems in consequential real-world settings. The methods could in principle be misused to automate test-taking, but the scope of this paper is limited to research on mathematical reasoning benchmarks.

Intended use of released artifacts. Any code or data released alongside this paper is intended for academic research on language model evaluation and mathematical reasoning. It should not be used outside of research contexts.

Use of AI assistants. The authors used AI writing assistants (including large language models) to assist with drafting and editing portions of this paper, and AI coding assistants to support code development. All content was reviewed, verified, and revised by the authors, who take full responsibility for the accuracy of the paper.

B Full Ranking Quality Figures

Figure 4 shows the full unfiltered version of Figure 1, including all questions regardless of oracle accuracy.

C Prompts

C.1 Proposal Prompt

We sample candidate responses from p_{LM} using the following system prompt:

```
You are a math assistant. Solve the problem
step-by-step and provide your final answer in
LaTeX format, ensuring the final result is
placed inside \boxed{}.
```

For **base models**, the prompt is formatted as a plain string:

```
{system_prompt}
Question: {question}
Answer:
```

For **chat models**, the prompt is formatted as a conversation:

```
[
  {
    "role": "system",
    "content": {system_prompt}
  },
  {
    "role": "user",
    "content": {question}
  }
]
```

C.2 Self-Verification Prompt

For **base models**, the full verification prompt $v(x, y)$ has the following structure:

```
{Instruction}
{Grading rules}
### Problem: {x}
### Solution: {y}
Take a moment to evaluate the logic
internally. {Virtual thinking filler}
### Verdict:
```

For the **chat models**, the full $v(x, y)$ in formatted as a conversation:

```
[
  {
    "role": "user",
    "content": (
      "{Instruction}"
      "{Grading rules}"
      "### Problem: {x}"
      "### Solution: {y}"
      "Take a moment to evaluate the logic
internally."
    )
  },
  {
    "role": "assistant",
    "reasoning_content": "{Virtual thinking
filler}",
    "content": "### Verdict:"
  }
]
```

We add one sentence to elicit virtual thinking across all settings, including the no-filler (normal) condition. By terminating the prompt with the incomplete string "`### Verdict:`", the model's next

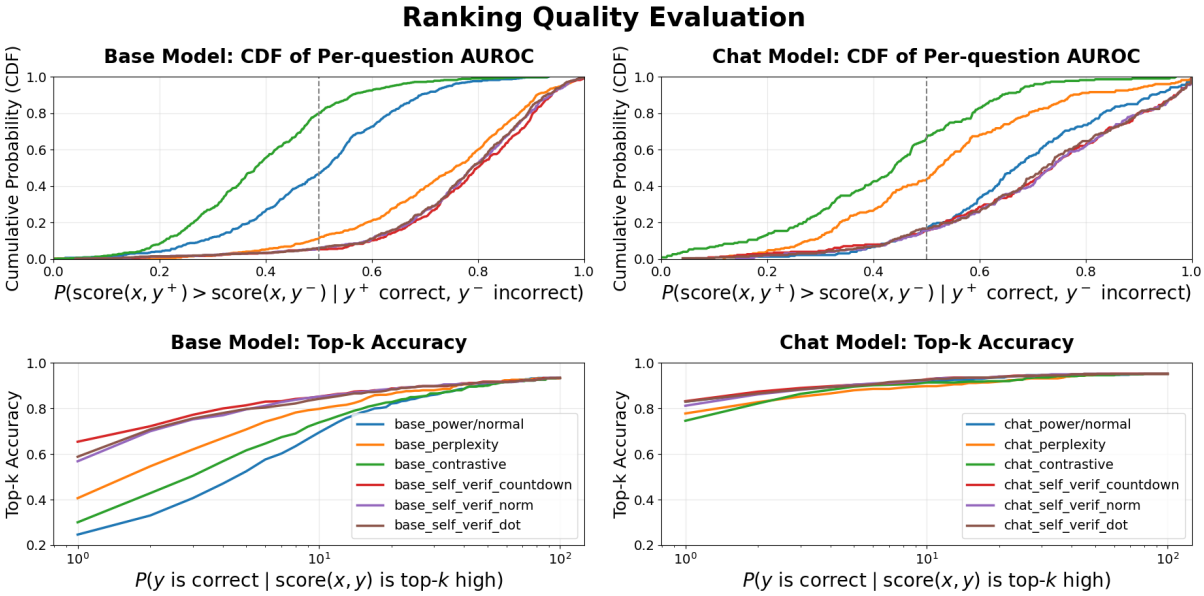

Figure 4: Full ranking quality of different scores for the Base (left) and Chat (right) models, including all questions (unfiltered). *Top:* CDF of per-question AUROC over questions with valid pools. *Bottom:* Top-k accuracy on all questions.

token is constrained to be " ✓" or " ✗", whose log-probabilities feed directly into Eq. (1).[7]

Instruction. The Instruction is as follows:

```
Given the math problem, review the following
solution carefully and determine if it is
correct.
- If the solution is correct, output
  "### Verdict: ✓".
- If the solution contains a logical error,
arithmetic error, or has wrong reasoning,
output
  "### Verdict: ✗
  ### Reason: . . . "
  with a brief explanation.
```

Grading Rules The grading rules for MCMC and candidate ranking are as follows:

```
GRADING RULES:
1. SCOPE: Judge both mathematical correctness
and answer formatting. A solution must be
mathematically sound AND produce a properly
formatted final answer to be marked ✓.
2. FINAL ANSWER REQUIRED: A final answer MUST
be present. It must appear inside \boxed{}
with no trailing punctuation or extra text
inside the box. If no \boxed{} answer is
present, or the box contains extra
text/punctuation, mark as ✗.
3. MATHEMATICAL CORRECTNESS: If the solution
contains any logical error, arithmetic error,
or invalid reasoning step that affects the
final answer, mark as ✗.
4. NO PROGRESS / LOOPING: If the solution
repeats the same step or sequence of steps
three or more times without producing new
intermediate results or progressing toward a
final answer, mark as ✗.
5. NOTATION TOLERANCE: Minor notational or
spacing differences (e.g. '1/2' vs
'\frac{1}{2}', extra whitespace, equivalent
algebraic forms) are not errors.
CRITICAL INSTRUCTION: Before outputting the
'### Verdict:' line, you must carefully think
through your evaluation internally as the
countdown progresses. Do not rush to give the
answer—explicitly engage in silent,
step-by-step reasoning during the countdown
before revealing your conclusion.
```

For SMC which requires scoring over partial solution sequence, the grading rules are as follows:

[7] In our implementation using the qwen tokenizer, the verdict tokens correspond to the two-token sequences " ✓" → (25521, 227) and " ✗" → (51414, 234). We take $p_{✓}$ and $p_{✗}$ to be the probability of the *first two* tokens in each sequence.

```
GRADING RULES:
1. A partial solution with no final answer
yet should be judged ONLY on whether its
steps and reasoning are correct so far. Do
NOT penalize for being incomplete.
2. If the solution is overly repetitive or
cycling through the same steps without making
progress, mark as ✗.
3. If the solution terminates early (e.g.
with an end-of-text token) before any
boxed answer appears, mark as ✗as the model
stopped without providing a final answer.
4. If a final answer IS present, it must
appear in
boxed with no trailing punctuation or extra
text inside the box, otherwise it is ✗.
5. Judge mathematical correctness only. Minor
notational or spacing differences are not
errors.
CRITICAL INSTRUCTION: Before outputting the
'### Verdict:' line, you must carefully think
through your evaluation internally as the
countdown progresses. Do not rush to give the
answer—explicitly engage in silent,
step-by-step reasoning during the countdown
before revealing your conclusion.
```

Virtual thinking filler. We investigate two virtual thinking variants:

- Countdown filler.

  ```
  Counting down: $n, $(n-1), . . . , 2,
  1. . .
  ```

  where n is the countdown length. The decreasing integer sequence signals to the model that it should complete an internal reasoning process of commensurate depth before producing the verdict.

- Dot filler.

  ```
  . . . [d dots] . . .
  ```

  where d is the number of dot characters inserted. This variant is motivated by the observation that punctuation tokens can serve as implicit "thinking" tokens in autoregressive models (see, e.g., Goyal et al., 2024), providing a lightweight inductive bias for deliberation without the semantic content of a countdown.

Neither variant requires the model to generate additional tokens; both are part of the input context and do not affect the single-forward-pass evaluation cost of s_{sv}.

C.3 Empirical Validation of Verdict Token Probabilities

We verify that the score s_{sv} in Eq. (1) is well-defined by confirming that the verdict tokens ✓ and ✗ together account for nearly all of the model's probability mass at the verdict position, i.e.

$$p_{\checkmark}(x, y) + p_{\times}(x, y) \approx 1 \tag{11}$$

holds empirically across inputs.

To validate Eq. (11), we compute $p_{\checkmark}(x, y) + p_{\times}(x, y)$ for candidates y in **Normal pool** using **Qwen3-1.7B**. Table 2 reports the mean, median, min, max, and 5th-percentile values of the summed probability across models on the Normal pool.

C.4 Contrastive Decoding: Structured Analysis Prompt

For base models, the expert distribution in contrastive decoding is conditioned on a structured analysis prefix that decomposes the problem into type, tools, key quantities, traps, strategy, and a sanity check plan (truncated to 2,048 tokens). The analysis is elicited via in-context learning using the following prompt:

```
Q: If a train leaves Chicago at 9:00 AM
traveling at 60 mph, and another train leaves
New York (800 miles away) at 10:00 AM
traveling at 80 mph, at what time will they
meet?
Analysis:
Problem Type & Domain: This is a relative
motion / meeting-point problem. Domain:
elementary kinematics + algebra.
Mathematical Tools Required: linear equations,
distance-rate-time formula (d = r·t), system
of two equations in one unknown.
Key Quantities & Unknowns: Let t = hours
after 9:00 AM. Train 1 distance: 60t. Train 2
distance: 80(t−1). Unknown: t where distances
sum to 800.
Traps & Edge Cases: Train 2 has a 1-hour head
start offset — forgetting this is the #1
mistake. Units are consistent (miles, hours)
— no conversion needed.
Solution Strategy: Set up 60t + 80(t−1) =
800, solve for t, convert decimal hours to
clock time.
Sanity Check Plan: Plug t back in, verify
both distances sum to 800; check answer is
after 10:00 AM.
[END OF ANALYSIS]
—
Q: {question}
Analysis: Problem Type & Domain:
```

The model's continuation of the `Analysis:` field constitutes the structured prefix; generation is trun-

Model	Mean	Median	5th pct.	Min	Max
Base	0.9588	0.9639	0.9209	0.71788	1.0097
Chat	0.9995	0.9995	0.9990	0.9395	1.0566

Table 2: Summary statistics of $p_{\checkmark} + p_{\times}$ across models on the Normal pool. Values close to 1.0 confirm the validity of the odds-ratio formulation. Values marginally exceeding 1.0 are due to floating-point rounding in the probability computation.

cated at 2,048 tokens before being prepended to the answer prompt.

D Failure Modes of Score-Guided Inference

This appendix analyzes a fundamental failure mode shared by both the MCMC and SMC samplers introduced in Section 4.2, explains why the target distribution $p_{\text{final}}(y \mid x) \propto p_{\text{LM}}(y \mid x) \cdot s(y \mid x)$ is a principled choice in light of this failure mode, and derives the acceptance ratio under this target.

D.1 Theoretical Analysis

A unified diagnostic. Under our MCMC sampler with the prefix-resample proposal, a new candidate y' is generated by sampling a shared prefix y_{pre} from the current response y and continuing autoregressively. Conditioned on y_{pre}, the proposal distributions are $q(y' \mid y, y_{\text{pre}}) = p_{\text{LM}}(y'_{\text{suf}} \mid x, y_{\text{pre}})$ and $q(y \mid y', y_{\text{pre}}) = p_{\text{LM}}(y_{\text{suf}} \mid x, y_{\text{pre}})$, where $y_{\text{suf}}, y'_{\text{suf}}$ are the suffices satisfying $y = y_{\text{pre}} \| y_{\text{suf}}$ and $y' = y_{\text{pre}} \| y'_{\text{suf}}$, with $\|$ denoting string concatenation. The resulting Metropolis–Hastings acceptance ratio is

$$\begin{aligned}\alpha &= \min\left(1,\ \frac{p_{\text{LM}}(y_{\text{suf}} \mid x, y_{\text{pre}})\, p_{\text{final}}(y' \mid x)}{p_{\text{LM}}(y'_{\text{suf}} \mid x, y_{\text{pre}})\, p_{\text{final}}(y \mid x)}\right) \\ &= \min\left(1,\ \frac{p_{\text{LM}}(y \mid x)\, p_{\text{final}}(y' \mid x)}{p_{\text{LM}}(y' \mid x)\, p_{\text{final}}(y \mid x)}\right) \\ &= \min\left(1,\ \frac{p_{\text{final}}(y' \mid x)/p_{\text{LM}}(y' \mid x)}{p_{\text{final}}(y \mid x)/p_{\text{LM}}(y \mid x)}\right), \quad (12)\end{aligned}$$

where the second line follows from multiplying numerator and denominator by $p_{\text{LM}}(y_{\text{pre}} \mid x)$ and applying the chain rule $p_{\text{LM}}(y_{\text{pre}} \mid x)\, p_{\text{LM}}(y_{\text{suf}} \mid x, y_{\text{pre}}) = p_{\text{LM}}(y \mid x)$. If the target were $p_{\text{final}} \propto s$ alone (without the p_{LM} factor), the acceptance ratio would then be

$$\alpha = \min\left(1,\ \frac{s(y' \mid x)/p_{\text{LM}}(y' \mid x)}{s(y \mid x)/p_{\text{LM}}(y \mid x)}\right). \quad (13)$$

Similarly, the particle weights in our SMC sampler are proportional to $p_{\text{final}}(y \mid x)/p_{\text{LM}}(y \mid x)$; if the target were $p_{\text{final}} \propto s$ alone, the importance weights would then be $s(y \mid x)/p_{\text{LM}}(y \mid x)$. In both cases, the quantity

$$r(y \mid x) \ := \ \frac{s(y \mid x)}{p_{\text{LM}}(y \mid x)} \quad (14)$$

is the central object governing sampler behavior: it appears as the ratio in the MCMC acceptance decision (13), and it is precisely the importance weight that would arise in SMC if p_{LM} were absent from the target.

Degeneration when the score is flat. Suppose $s(y \mid x)$ has low dynamic range relative to $p_{\text{LM}}(y \mid x)$—that is, s varies slowly across responses while p_{LM} varies sharply. In this regime, the ratio in Eq. (13) reduces to

$$\alpha \approx \min\left(1,\ \frac{p_{\text{LM}}(y \mid x)}{p_{\text{LM}}(y' \mid x)}\right), \quad (15)$$

and the score ceases to guide the search. The same degeneration affects SMC—particle weights collapse to $1/p_{\text{LM}}(y \mid x)$, which favors longer responses regardless of their correctness, since p_{LM} assigns exponentially lower probability to longer sequences. This reveals a shared failure mode: *both MCMC and SMC fail to sample from p_{final} whenever the score lacks sufficient dynamic range relative to the language model.*

The power distribution as a structural fix. The only family of scoring functions that *structurally* avoids the above failure mode for all inputs is the power distribution,

$$p_{\text{final}}(y \mid x) \ \propto \ p_{\text{LM}}(y \mid x)^{\alpha}, \quad (16)$$

which corresponds to $s(y \mid x) = p_{\text{LM}}(y \mid x)^{\alpha}$. Under this choice, the diagnostic ratio (14) becomes $r(y \mid x) = p_{\text{LM}}(y \mid x)^{\alpha-1}$, which is itself a power of p_{LM} and hence always has well-matched dynamic range by construction. No task-motivated score—including the verification score from Section 3—can provide this guarantee in general: although such scores are derived from p_{LM}, the transformations involved (e.g., contrastive subtraction,

self-verification prompting, virtual thinking countdown) are not monotone rescalings of p_{LM}, and may therefore exhibit dynamic range mismatch on some inputs.

Pragmatic justification for $p_{\mathrm{final}} \propto p_{\mathrm{LM}} \cdot s$. While the power distribution offers a structural guarantee, it provides no mechanism for incorporating task-relevant signal beyond likelihood rescaling. Our choice of $p_{\mathrm{final}}(y \mid x) \propto p_{\mathrm{LM}}(y \mid x) \cdot s(y \mid x)$ is motivated as the principled middle ground:

- **Compared to $p_{\mathrm{final}} \propto s$ alone.** Including p_{LM} in the target ensures that the importance weights in SMC are proportional to $s(y \mid x)$ rather than $s(y \mid x)/p_{\mathrm{LM}}(y \mid x)$, directly suppressing the pathological $1/p_{\mathrm{LM}}$ term. The failure mode is reduced to cases where s is genuinely uninformative—a property of the score itself, not an artifact of the framework.

- **Compared to $p_{\mathrm{final}} \propto p_{\mathrm{LM}}^{\alpha}$.** Multiplying by a task-motivated score s injects correctness-oriented signal that temperature scaling alone cannot provide, enabling the sampler to concentrate on responses that are both fluent and factually grounded.

In summary, $p_{\mathrm{final}} \propto p_{\mathrm{LM}} \cdot s$ is the natural choice when s is a task-motivated score: it inherits the dynamic-range benefits of anchoring to p_{LM} while still allowing the score to reshape the distribution toward correct responses.

D.2 Empirical Illustration

To empirically verify the failure mode, we trace the MCMC chain and examine transitions where the score and the language model prior are in tension. Concretely, for each proposed transition $(y \to y')$, we define

$$\Delta_s = \log s(y' \mid x) - \log s(y \mid x), \tag{17}$$

$$\Delta_\pi = \log p_{\mathrm{LM}}(y' \mid x) - \log p_{\mathrm{LM}}(y \mid x). \tag{18}$$

Recall from Eq. (13) that the log acceptance ratio is $\Delta_s - \Delta_\pi$. A *conflict* arises when Δ_s and Δ_π share the same sign: both the score and the prior nominally prefer the same candidate, yet because p_{LM} appears in the denominator of the acceptance ratio, it actively resists the transition that the score favors. In such cases, the score *wins* if its signal dominates, i.e., $|\Delta_s| > |\Delta_\pi|$, ensuring $\Delta_s - \Delta_\pi$ and Δ_s share the same sign: both prefer the same candidate. We therefore compute two statistics over all conflicting transitions observed during the chain:

- **Conflict rate**: the fraction of all proposed transitions that are conflicts,

- **Score win rate**: among conflicts, the fraction where $|\Delta_s| > |\Delta_\pi|$.

A high conflict rate paired with a low score win rate is the empirical signature of the failure mode: the prior dominates the acceptance decision, and the chain reduces to exploring p_{LM} rather than p_{final}.

Table 3 reports these statistics for each scoring function considered in this work.

The results confirm the theoretical analysis. The power distribution achieves a conflict rate and win rate of 1.0000 by construction, serving as a sanity check: since $s(y \mid x) = p_{\mathrm{LM}}(y \mid x)^{\alpha}$, the diagnostic ratio becomes $r(y \mid x) = p_{\mathrm{LM}}(y \mid x)^{\alpha-1}$, which amplifies the dynamic range of p_{LM} and ensures the score always dominates by algebraic guarantee. Perplexity exhibits the most severe failure: despite a high conflict rate of 0.6538, its win rate is effectively zero (0.0003), meaning the prior overwhelmingly dominates every contested transition. This is expected: perplexity normalizes $\log p_{\mathrm{LM}}(y \mid x)$ by sequence length $|y|$, which *flattens* the dynamic range of p_{LM} rather than amplifying it—the opposite effect to the power distribution—leaving p_{LM} in full control. Self-verification, while exhibiting a more moderate conflict rate (0.4626), similarly suffers from a near-zero win rate (0.0411), indicating that its dynamic range remains insufficient relative to p_{LM} despite being a more informed score. Contrastive decoding occupies a middle ground with a win rate of 0.3282, suggesting that the contrastive formulation incidentally produces dynamic range more comparable to p_{LM}—though still far from the structural guarantee of the power distribution. Taken together, these results underscore that the failure mode is not merely a theoretical concern but a measurable phenomenon, and motivate the use of $p_{\mathrm{final}} \propto p_{\mathrm{LM}} \cdot s$ as a principled mitigation that anchors the target distribution to p_{LM}.

E Self-Verification Prompt Design Ablations

We ablate the self-verification (virtual-thinking) prompt along three axes: the *filler length*, the *filler type* (a repeated dot sequence vs. a descending

Score	Conflict Rate	Win Rate
Power Distribution Likelihood	1.0000	1.0000
Perplexity	0.6538	0.0003
Contrastive Decoding	0.3004	0.3282
Self-Verification (ours)	0.4626	0.0411

Table 3: Conflict rate and score win rate for each scoring function under the MCMC sampler. A high conflict rate with a low win rate indicates that p_{LM} dominates the acceptance decision.

countdown sequence), and—on the Chat model—the *placement* of the prompt within the chat template. We further test whether any of these choices transfers to consensus accuracy. All ablations use the same generation pool of $N{=}100$ candidates per question on MATH500, so that differences reflect the scoring prompt alone and not variation in the candidate set. We report three metrics. **Top-k accuracy**—the fraction of questions for which at least one correct candidate appears among the top-k scored—captures precision-at-one quality relevant to best-of-N selection. **Mean per-question Spearman correlation**—the Spearman rank correlation between score and binary correctness computed within each question and averaged over questions with valid pools (those containing at least one correct and one incorrect candidate)—measures overall within-question ordering. **Weighted-consensus accuracy**—the fraction of questions whose score-weighted plurality answer is correct—measures the downstream effect when the score is used to reweight a majority vote rather than to select a single candidate.

E.1 Chat Model Virtual Thinking Behavior

Settings. We examine how the Chat model responds to virtual thinking compared to the Base model, using the same length sweep over $n \in \{10, 20, 30, 40, 50, 60, 70, 80\}$ countdown tokens with the no-filler baseline as reference (Figure 6, right panel). We additionally compare dot and countdown fillers at matched lengths to assess whether filler type interacts with instruction tuning.

Results. The Chat model behaves qualitatively differently from the Base model under length variation. Whereas the Base model exhibits an approximate unimodal pattern in Top-k accuracy at small k, no consistent trend emerges on the Chat model: curves at different lengths interleave without a clear ordering, and mean per-question Spearman correlation likewise shows no monotone relationship with n. Despite the absence of a length trend, virtual thinking remains beneficial: the no-filler baseline (`no_count`) sits at or near the bottom of the Chat panel across all k, confirming that inserting *some* filler before the verdict improves discrimination even when the optimal length is unclear. A second difference from the Base model is that the dot filler occasionally outperforms the countdown filler on Chat, whereas countdown consistently leads on Base. We speculate that the Chat model, having been trained on structured instruction-following data, is less sensitive to the semantic content of the filler sequence and responds roughly equally to any non-empty occupancy of the pre-verdict context window.

Prompt placement. Because the Chat model uses a structured chat template, the self-verification filler can be inserted in different turns: inside the assistant's reasoning span (*think*), as plain assistant text (*assist*), or in the user turn (*user*). Figure 5 sweeps these placements for both filler types. All six placement×type variants lift Top-k accuracy at small k over the default no-filler prompt (`normal`), confirming once more that some virtual thinking helps. But the gains are small, the curves converge by moderate k, and no single placement dominates: the *think* placements are marginally ahead at $k{=}1$, yet the ordering is not stable across k. Moreover, the placement variants show no gain in mean per-question Spearman correlation over the default prompt (Table 4, bottom block), so the effect, such as it is, is confined to precision-at-one. Placement thus behaves like the other Chat-model design axes—a small, inconsistent top-of-ranking effect with no change in overall ordering.

E.2 Effect of Length

Settings. We sweep the countdown filler length over $n \in \{10, 20, 30, 40, 50, 60, 70, 80\}$ tokens, dot filler length of $d \in \{10, 20, 50\}$ with the no-

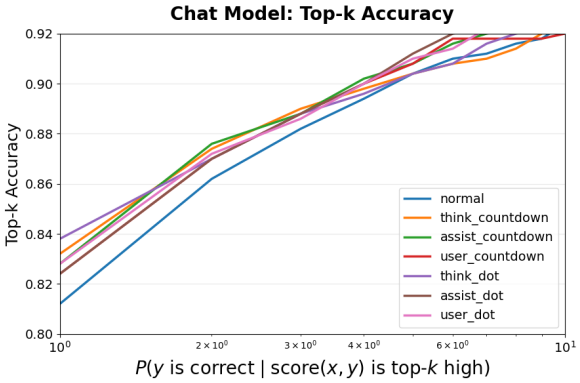

Figure 5: **Effect of self-verification prompt placement on the Chat model.** Top-k accuracy for the six placement×filler-type variants—{think, assist, user}×{countdown, dot}—against the default no-filler prompt (`normal`). Every placement lifts Top-k accuracy at small k, but the gains are small and no placement is consistently best.

filler baseline (SV-none) included as a reference. Longer fillers give the model more context-window occupancy before the verdict, potentially allowing a more calibrated internal state, at the cost of additional compute per scoring call. We report Top-k accuracy as a function of k on both the Base and Chat models.

Results. Figure 6 shows Top-k accuracy for each countdown length on the Base and Chat models. On the Base model, the relationship between length and performance is approximately unimodal at small k: accuracy rises from the no-filler baseline and peaks around n=40–50 before declining at longer lengths, consistent with a deliberation-budget interpretation in which a moderate amount of context occupancy sharpens the verdict without over-saturating the model's attention. The exception is n=10, which at small k performs notably better than its position in the monotone region would suggest—above n=20 and n=30. We attribute this to the frequency of the countdown sequence "$10, 9, \ldots, 1$" in pretraining corpora, where such sequences appear in everyday text (countdowns, lists, timers), making this particular filler a familiar, semantically loaded context rather than a neutral occupancy token; the model may therefore respond to it with especially well-calibrated internal state. Both effects diminish as k grows and the curves converge, confirming that virtual thinking's benefit is concentrated at the top of the ranking. On the Chat model, no consistent ordering emerges across lengths: the curves interleave throughout the k range, suggesting that the instruction-tuned model's output distribution is already well-concentrated and less sensitive to the precise form of the virtual-thinking filler.

Table 4 reports the mean per-question Spearman correlation for all variants. On the Base model, all countdown lengths improve substantially over the no-filler and norm baselines (≈.355), clustering tightly between .362 and .366 with no monotone trend—the spread across lengths is smaller than the gap to the baseline, suggesting that *any* countdown length is effective and the precise value matters little for overall ranking quality. Dot fillers do not improve over the no-filler baseline on Base and are marginally below norm. On the Chat model, correlations are uniformly low (≈.21) across all variants with no consistent ordering, consistent with the Top-k finding.

Dot filler. Figure 7 repeats the length sweep for the dot filler. On the Base model the dot curves lie modestly above the no-filler baseline at small k, but the separation is smaller than for the countdown filler (cf. Figure 6) and is not ordered by length; in mean per-question Spearman correlation the dot variants are statistically indistinguishable from the no-filler baseline (Table 4). The countdown filler thus improves *both* metrics, whereas the dot filler yields at most a weak Top-k effect and no gain in overall ordering—isolating the *semantic content* of the countdown sequence, rather than filler length or mere context occupancy, as the active ingredient on the Base model. On the Chat model the dot curves interleave with the baseline, as for the countdown filler.

On adaptive filler length. The results above fix a single length for all questions. A natural extension is a *problem-adaptive* budget: map each question x to a filler length $L(x)$ based on an estimate of its difficulty, allocating more deliberation to harder questions. One simple proxy is the oracle accuracy of the candidate pool, though this is not available at test time; a practical surrogate might be the entropy of the score distribution over the pool. We leave a full evaluation of adaptive-length strategies to future work.

E.3 Effects on Consensus

Settings. The main text notes that virtual thinking barely affects consensus accuracy—the dot and countdown fillers are near-indistinguishable

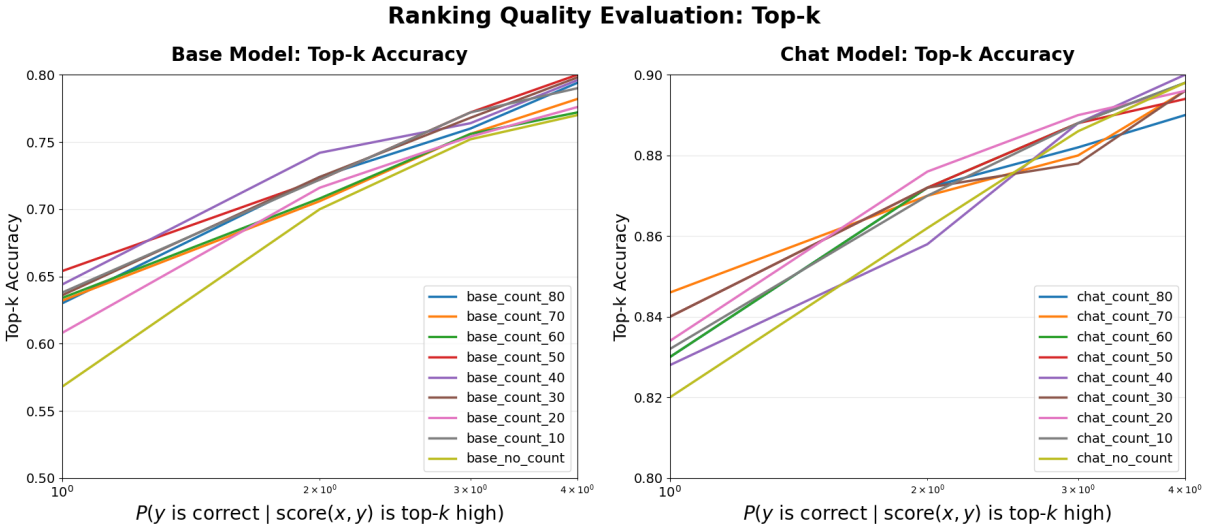

Figure 6: **Effect of countdown filler length on Top-k accuracy.** Each curve corresponds to one countdown length $L \in \{10, 20, 30, 40, 50, 60, 70, 80\}$; the no-filler baseline (`no_count`) is shown in yellow-green for reference. *Left*: Base model. *Right*: Chat model. On the Base model performance follows an approximate unimodal pattern as a function of L, peaking around L=40–50, with L=10 as a notable exception (see text). On the Chat model no consistent trend emerges.

when s_{sv} is used as a reweighting signal for majority vote. Here we confirm this finding across the full ablation grid. For every variant we compute weighted-consensus accuracy: for each question we sum s_{sv} over the candidates proposing each distinct answer and take the highest-weighted answer as the consensus prediction. We report this for the no-filler self-verification prompt, the dot filler at $d \in \{10, 20, 50\}$, the countdown filler at $n \in \{10, \ldots, 80\}$, and—on the Chat model—the six placement variants. As a reference we include `normal`, the unweighted (uniform-vote) majority baseline that uses no self-verification signal; because all variants reweight the same shared generation pool, `normal` is identical for the Base and Chat scorers.

Results. Figure 8 reports weighted-consensus accuracy for every variant. The dominant effect is the presence of the self-verification signal itself: reweighting the plurality vote by s_{sv} raises consensus accuracy from the unweighted baseline of .698 to ≈.75 on the Base scorer and to ≈.87 on the Chat scorer. Conditioned on using a self-verification prompt, however, none of the design axes moves the needle: the Base variants span only .750–.758 and the Chat variants only .870–.878, a range of .008 in both cases, and the ordering within that band is inconsistent with the length, type, and placement trends seen for ranking quality. In particular, the cheap no-filler prompt (.758 on Base, .874 on Chat) already sits at the top of its cluster, tied with the best countdown and placement variants.

Notably, the countdown filler's advantage on the Base model—clear in Top-k and Spearman—does *not* carry over to consensus: the best countdown variants (.758) only match the no-filler prompt. This is expected. Consensus aggregates over the entire pool and is determined by which *answer* accumulates the most weight, so it is robust to small perturbations in the score of any individual candidate; the gains from a better filler are concentrated at the very top of the ranking, exactly the regime that best-of-N selection—but not plurality voting—exploits. Self-verification prompt design is therefore a selection-time lever, not a consensus-time one.

E.4 Summary

Across all three ablation axes the picture is consistent. The decision that matters is binary—whether to use a self-verification signal at all. Adding one produces large gains in every metric (Top-k, Spearman, and consensus) over the `normal` baseline; conditioned on using one, the prompt's exact form is close to a free parameter. No design choice yields a statistically meaningful difference in consensus accuracy (all variants fall within a .008 band), and on the Chat model no choice yields a meaningful difference in ranking quality either—Top-k, Spearman, and consensus curves all interleave without a

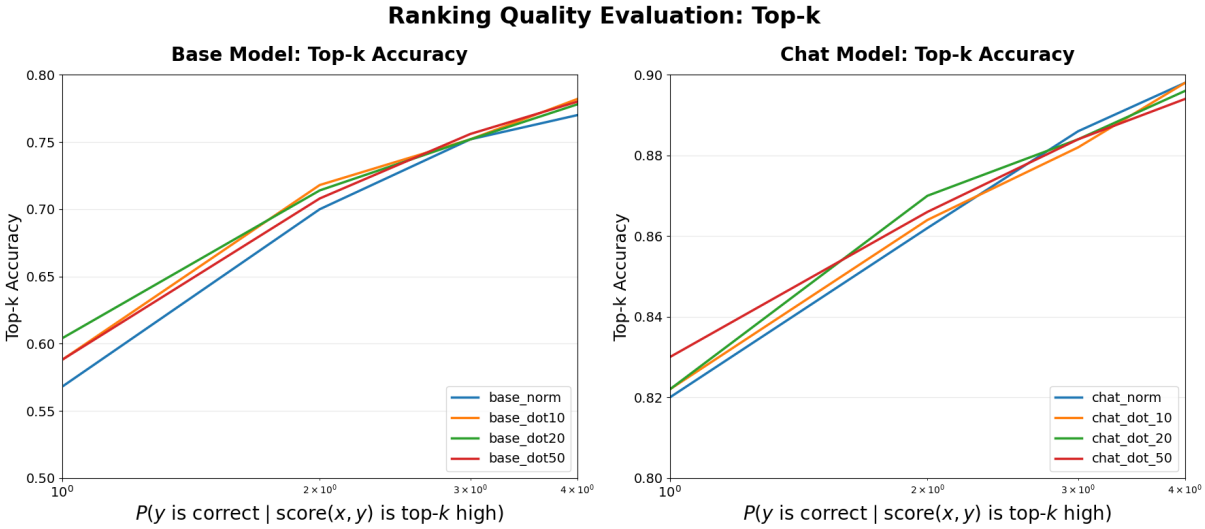

Figure 7: **Effect of dot-filler length on Top-k accuracy.** Length sweep for the dot filler, $L \in \{10, 20, 50\}$, with the no-filler baseline (`norm`) for reference. *Left*: Base model. *Right*: Chat model. Unlike the countdown filler (Figure 6), the dot filler produces only a small Top-k separation from the baseline on the Base model and none on the Chat model, and shows no ordering by length.

stable order.

The one real—if modest—difference appears in best-of-N *selection* on the Base model. There the countdown filler improves both Top-k accuracy at small k and mean per-question Spearman correlation ($\approx$.365 vs. $\approx$.355 for the no-filler and dot prompts), consistently across all swept lengths, whereas the dot filler does not improve overall ordering. This advantage is specific to the top of the ranking and does not transfer to consensus, where every variant is tied.

We draw three design recommendations. For best-of-N selection with a Base-style model, use a countdown filler; the precise length is not critical, as any value in the swept range works, with a mild preference for n=40–50. For a Chat-style instruction-tuned model the filler form is immaterial—dot, countdown, and placement are interchangeable—so the cheapest option, a short or no-filler prompt, is preferred. And when the downstream goal is consensus or weighted majority vote rather than single-candidate selection, self-verification prompt design can be ignored entirely: all variants, including the inexpensive no-filler prompt, are statistically indistinguishable, so no tuning is warranted.

F Scoring Functions as Confidence for Abstention

Ranking candidates *within* a question and deciding *whether to answer at all* make different demands on a score. Ranking needs only the correct *ordering* of candidates inside each question; abstention needs more—scores must be comparable *across* questions, so that a single global threshold can separate the questions a model should answer from those it should decline. Whether a score meets this stronger requirement depends on its *calibration scope*: the range over which its magnitudes carry a consistent meaning. Calibration scope is set by the score's functional form, not by model quality, and it differs from one scoring function to the next. We therefore study all four scoring functions side by side as confidence signals for abstention, asking how far each one's calibration scope extends and what limits it.

Settings. We pool all $N \times |\mathcal{X}|$ (question, candidate) pairs and report the **Spearman rank correlation** between the score $s(y \mid x)$ and binary correctness $c(y)$ over the pooled pairs. Pooling is what turns this into a test of calibration scope: a score that orders candidates well *within* each question but whose magnitudes are not comparable across questions will rank a confident wrong candidate from one question above a less-confident correct candidate from another, depressing the pooled correlation. Because Spearman depends only on the score ordering, it is invariant to any monotone rescaling and places all four scoring functions on equal footing. To probe difficulty-conditioned discrimination we recompute it on subsets filtered by per-question oracle accuracy: the "<0.90", "<0.80",

Variant	Base	Chat
Baseline		
No filler	.3552	.2115
Norm	.3552	.2115
Dot filler		
Dot-10	.3525	.2121
Dot-20	.3559	.2115
Dot-50	.3520	.2106
Countdown filler		
Count-10	.3654	.2054
Count-20	.3645	**.2141**
Count-30	**.3659**	.2077
Count-40	.3643	.2065
Count-50	.3650	.2113
Count-60	.3625	.2025
Count-70	.3639	.2108
Count-80	.3652	.2061
Chat prompt placement (Chat only)		
Normal	—	.2133
Think-countdown	—	.2131
Assist-countdown	—	.2094
User-countdown	—	.2047
Think-dot	—	.2101
Assist-dot	—	.2091
User-dot	—	.2109

Table 4: **Mean per-question Spearman rank correlation** for all self-verification variants, grouped by filler type (baseline, dot, countdown) and, for the Chat model, by prompt placement. Bold marks the best value per column within the length-ablation groups. On the Base model, all countdown lengths yield a substantial improvement over the no-filler baseline with no clear ordering among lengths; dot fillers do not improve over baseline. On the Chat model, all variants cluster near .21 with no consistent trend. The prompt-placement block (bottom) is Chat-only; "Think-countdown" is identical to Count-50.

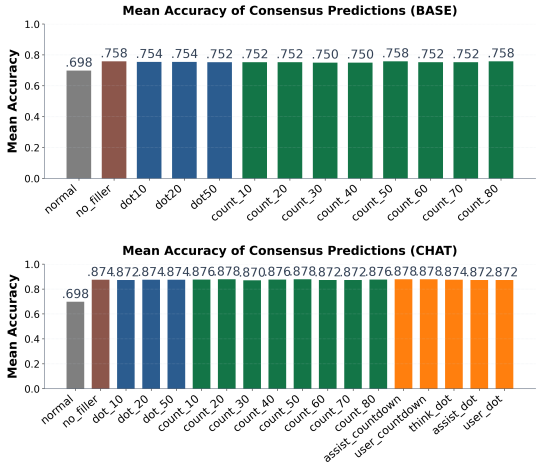

Figure 8: **Weighted-consensus accuracy across all self-verification variants.** Each bar is the fraction of MATH500 questions whose score-weighted plurality answer is correct, over the shared $N{=}100$-candidate pool. *Top*: Base scorer. *Bottom*: Chat scorer. The leftmost bar (`normal`) is the unweighted majority-vote baseline and uses no self-verification signal. Reweighting by s_{sv} lifts consensus sharply ($.698 \rightarrow \approx .75$ on Base, $.698 \rightarrow \approx .87$ on Chat), but among the self-verification variants—across filler length, filler type, and prompt placement—accuracy varies by at most .008 on either model, with no consistent ordering.

and "<0.50" columns progressively retain only harder questions, whose candidate pools contain fewer correct answers and on which an abstention decision is most consequential.

Results. Table 5 reports the Spearman rank correlation under four difficulty thresholds, and the scores separate cleanly by calibration scope.

Self-verification has the widest scope. s_{sv} is the normalized probability of a single verdict token—an estimate of $P(\text{correct})$ on a fixed $[0, 1]$ scale, computed identically for every question and free of the mechanical length penalty that afflicts likelihood products. Its magnitudes are comparable across questions by construction, and this is visible as robustness: SV-countdown leads every column on the Base model, and all three SV variants retain a substantial pooled correlation ($\approx .27$) even on the hardest subset (<0.50), after the easy questions that inflate other scores have been filtered away.

The likelihood-derived scores are calibrated, at best, only *within* a question, and each is bounded by a different artifact of its functional form. **Power**, $p_{\mathrm{LM}}(y \mid x)^{\beta}$, applies no length normalization: sequence probability decays geometrically in $|y|$ and the exponent $\beta{=}4$ sharpens the decay, so

	Base				Chat			
Score	All	$<.90$	$<.80$	$<.50$	All	$<.90$	$<.80$	$<.50$
Power	.338	.305	.247	.111	**.493**	.301	.236	.154
Perplexity	.252	.250	.239	.123	$-.148$	$-.119$	$-.042$	$-.012$
Contrastive	.036	.006	$-.024$	$-.042$	$-.323$	$-.091$	$-.032$	$-.020$
SV-countdown	**.510**	**.497**	**.443**	**.275**	.472	.323	.280	.217
SV-norm	.483	.474	.425	.267	.474	.312	.274	**.219**
SV-dot	.489	.479	.428	.266	.479	**.325**	**.281**	.218

Table 5: **Spearman rank correlation with correctness** under four difficulty thresholds ("All" = no filter; <0.90/<0.80/<0.50 retain questions whose oracle accuracy falls below the threshold). Bold marks the best score per column. Negative values indicate anti-correlation. Chat-power leads on easy questions but collapses under thresholding; self-verification variants are the most robust across difficulty levels.

power rewards short responses regardless of correctness. Pooled across questions this length bias dominates—power's high unfiltered correlation on the Chat model ($.493$) is borrowed from the easy regime, where the model is confident and its probability mass genuinely tracks correctness; once easy questions are removed, the length and predictability confound is all that remains and the correlation collapses to $.236$ at <0.80 and $.154$ at <0.50, below every SV variant. **Perplexity**, $p_{\mathrm{LM}}(y \mid x)^{1/|y|}$, is length-normalized and so escapes the short-response bias, but the quantity it measures—average per-token predictability—is fluency, not correctness; across questions a generic or formulaic answer scores high whether or not it is right. This leaves a weak positive signal on the Base model and an actively misleading one on the Chat model, where the pooled correlation is negative throughout ($-.148$ on "All") because the instruction-tuned model emits fluent, confident-sounding wrong answers that perplexity rates highly. **Contrastive**, the expert/amateur likelihood ratio, cancels length to first order—both terms are products over the same $|y|$ tokens—but its magnitude tracks the expert/amateur gap, which varies per question and, on the Base model, depends on an engineered analysis prefix; it carries almost no pooled signal on Base ($.036$) and an anti-correlated one on Chat ($-.323$). The pattern is consistent: only self-verification has a calibration scope wide enough for cross-question abstention, while the likelihood scores are within-question instruments whose magnitudes, read across questions, import whichever artifact—response length, fluency, or a question-dependent baseline—their functional form fails to cancel.