Survey on Verifier-Guided Multi-Step Reasoning: A Review of Recent Advances

Ali Heydari *ali.heydari.1381@sharif.edu*
Sharif University of Technology

Mohammadjavad Mohammadi *mojavad.mohammadi95@sharif.edu*
Sharif University of Technology

Ali Ghanbarian *ali.ghanbarian1378@sharif.edu*
Sharif University of Technology

Radin Cheraghi *radinch1783@gmail.com*
Sharif University of Technology

Sahand Akramipour *sahandap@gmail.com*
Sharif University of Technology

Danial Parnian *danielparnian@gmail.com*
Sharif University of Technology

Abstract

Large Language Models (LLMs) increasingly solve problems that require extended chains of intermediate decisions, yet a correct final answer does not guarantee that the underlying reasoning is valid, robust to earlier mistakes, or safe to optimize. Verifier-guided reasoning addresses this gap by assessing complete traces, individual steps, or transitions between reasoning states and using the resulting signal for candidate selection, search, refinement, data construction, or policy optimization. This survey organizes the literature through a single six-category, contribution-oriented taxonomy: methodological refinements to search and process-reward objectives; critical analyses and failure modes; benchmarks and diagnostic evaluation; alternative supervision and verifier formulations; critique-based and domain-grounded verification; and efficiency-oriented methods. Within this taxonomy, we also examine the functional role that each verifier signal plays in the reasoning system, without treating that analytical lens as a second classification. We synthesize human and automated process supervision, outcome-to-process conversion, preference and value modeling, generative critique, latent verification, tool- and rule-grounded rewards, and adaptive test-time scaling. Empirical findings are compared only where experimental conditions are sufficiently aligned, with inference, training, annotation, and sample efficiency kept distinct. The resulting synthesis exposes recurring weaknesses—including miscalibration, reward hacking, search degradation, first-error truncation, distribution shift, and mismatch between reward semantics and downstream use—and motivates research on calibrated uncertainty, recovery-aware supervision, adaptive compute allocation, grounded evidence, standardized cost reporting, and safe self-improvement.

1 Introduction

Large Language Models (LLMs) have made multi-step reasoning a central capability of modern language-model research, with gains reported on mathematical problem solving, code generation, planning, and other tasks in which an answer depends on a sequence of intermediate decisions. The central reliability problem is that a plausible final answer does not imply a valid reasoning process: an early arithmetic error, unsupported inference, or invalid transformation can contaminate later steps, while a model may occasionally recover from an error and still reach a correct conclusion. Consequently, evaluating only the final answer provides an incomplete picture of reasoning quality.

Verifier-guided reasoning addresses this gap by introducing an additional signal that evaluates candidate reasoning traces or their intermediate steps. Depending on the system, the signal may be supplied by human process labels, an outcome or process reward model, a critique model, a tree-search value, an implicit reward, a latent-state statistic, an executable tool, or a deterministic domain rule. The verifier can then be used at inference time to rerank candidates, guide beam or Monte Carlo Tree Search (MCTS), trigger revision, or allocate additional test-time compute; at training time it can construct data, provide dense rewards, or support reinforcement learning. This makes verification a system-level design problem rather than a single model-architecture choice.

Adjacent surveys on test-time scaling and reasoning models provide useful broader context, but verifier-guided reasoning still lacks a consolidated account of the main forms of technical contribution, the function of the verification signal within the overall system, and the failure modes that emerge when the signal is scaled or optimized [1, 2]. The literature has become difficult to compare because the phrase "process reward" can denote local step correctness, future solvability, Q-value, progress, preference, constraint satisfaction, or recovery from a previous error. These signals are not interchangeable, and a verifier that is effective for best-of-N selection may be poorly suited to first-error diagnosis or reinforcement learning. Moreover, reported gains are often obtained under different generators, candidate budgets, verifier sizes, datasets, and evaluation protocols. A rigorous survey therefore needs both a stable taxonomy of the literature and an explicit analysis of *what is being verified*, *where the evidence comes from*, *how the judgment is represented*, *how the signal is consumed*, and *how the resulting system is evaluated*.

1.1 Why this survey is needed

This survey focuses on the transition from classical outcome-based selection toward process-aware, search-aware, and increasingly generative verification. Three developments make a consolidated view particularly useful. First, scalable supervision has moved beyond expensive human step labels toward rollout-based, confidence-based, active, preference-based, and implicit signals. Second, the verifier itself is becoming a reasoning agent through generative critique, long-chain verification, adaptive fast–slow judging, and self-verification. Third, recent analyses challenge the assumption that more verifier-guided search is always better: imperfect verifiers can prune correct branches, become exploitable rewards, or lose validity when a reasoning trace contains a genuine recovery after an error.

1.2 Contributions

The survey makes four concrete contributions:

1. **A unified six-category taxonomy.** We organize the literature by its primary technical contribution: methodological refinements; critical analyses and failure modes; benchmarks and diagnostics; alternative verifier formulations; critique-based and grounded verification; and efficiency-oriented methods. The categories provide one consistent structure for the review while allowing related papers to be examined through a complementary functional lens.

2. **A failure-mode synthesis.** We connect verifier failures across inference, diagnosis, and reinforcement learning, including misranking, reward hacking, calibration error, first-error truncation, distribution shift, and compute-induced degradation.

3. **A matched comparative synthesis.** Where the reviewed studies expose sufficiently comparable settings, we distinguish direct matched comparisons from contextual results reported across studies and explicitly separate inference, training, annotation, and sample efficiency.

4. **A research agenda.** We identify concrete gaps in calibrated supervision, adaptive compute allocation, recovery-aware semantics, grounded verification, benchmark validity, and safe self-improvement, and translate them into design directions.

1.3 Scope and review questions

The object of this review is *verifier-guided multi-step reasoning in LLMs*: systems in which an auxiliary judgment about a complete solution, an intermediate step, or a transition between reasoning states materially influences generation, selection, learning, or evaluation. A work is included when verification is a substantive technical component rather than a reporting metric or incidental prompt. Accordingly, the survey covers learned outcome and process reward models, human and automated process supervision, value- and preference-based verifiers, generative critique, implicit and latent verification, tool- or rule-grounded rewards, verifier-guided search, adaptive test-time computation, and the use of verification signals for data construction, self-training, or reinforcement learning. It also includes theoretical analyses and diagnostic benchmarks when their main purpose is to characterize the validity, scalability, or failure modes of such systems.

Mathematical reasoning is the dominant empirical setting in the reviewed literature because it provides exact final-answer checks and relatively structured intermediate steps. The scope nevertheless extends to code-like executable checks, structured domain decisions, and general critique whenever the method directly addresses multi-step verification. This broader inclusion is important for distinguishing methods that depend on mathematical answer checking from those that may transfer to domains with weaker or differently structured ground truth.

The survey does *not* attempt to cover the full literature on chain-of-thought prompting, self-consistency, tree search, RLHF, formal verification, theorem proving, or general LLM alignment. Such work is included only when it introduces, evaluates, or directly supports a verifier for multi-step reasoning. Pure model-scaling or prompting methods without an explicit verification signal, generic reward-modeling studies unrelated to intermediate reasoning, and application papers that merely report final-answer accuracy are outside the main scope. Outcome-only verification is discussed when it serves as a baseline, a source of process supervision, or a theoretically relevant bridge to process verification. Likewise, external tools and formal rules are considered only when they are integrated into an LLM reasoning or learning pipeline. Multimodal, embodied, and interactive-agent verification are not treated comprehensively because the reviewed corpus does not provide sufficient common evidence for a reliable synthesis.

The scope is organized around five review questions:

1. **RQ1 – Evidence:** What evidence is used to decide whether a reasoning step or trace is good?

2. **RQ2 – Semantics:** Does the verifier represent local correctness, recoverability, progress, value, preference, critique, constraints, or another quantity?

3. **RQ3 – Consumption:** How is the signal consumed by reranking, search, refinement, self-training, or policy optimization?

4. **RQ4 – Reliability:** Which failure modes appear when the verifier is imperfect, the generator changes, the trace contains recovery, or test-time compute grows?

5. **RQ5 – Efficiency and evaluation:** What should be measured to compare verifier quality, annotation cost, inference cost, and downstream usefulness fairly?

The review is therefore *structured and corpus-bounded rather than exhaustive.* It synthesizes the selected papers through a consistent taxonomy and reports numerical findings only when they are stated in the

reviewed material. Because experimental settings vary substantially and no pooled effect size is estimated, the quantitative section should be read as a comparative synthesis with explicitly matched subsets, not as a systematic meta-analysis.

2 Background

This section introduces the fundamental concepts, definitions, and notation necessary to understand verifier-guided reasoning. The terminology is intentionally explicit because many apparent disagreements in the literature arise from using the same word for different reward semantics.

2.1 Multi-Step Scaling

In multi-step reasoning tasks, a model generates a solution consisting of a sequence of intermediate steps. More formally, given a problem q and a **policy model** π, we generate a reasoning trace $(s_1, s_2, \ldots, s_K)$ where each s_k represents a reasoning step (e.g., a sentence or paragraph). The final answer is derived from the complete trace.

A primary approach is **Chain-of-Thought (CoT)** prompting, which encourages models to produce step-by-step reasoning before outputting the final answer. Later approaches moved towards training the model to produce CoT reasoning answers, instead of prompting the model to do so. More abstractly, at each step, the model chooses from a multitude of next-step options, making the problem similar to a tree search. A single CoT is therefore often called a "reasoning trace" or a "reasoning path". Tree-of-Thought (ToT) frameworks are based on this idea.

Another approach is **self-reflection**, in which a model critiques a completed response and uses the feedback to revise it. The critic may be the same model or a separate **critique model**. Recent work moves this feedback loop from prompting into training, making critique and verification explicit learned behaviors. The key limitation is that self-generated feedback is useful only when the verifier is sufficiently reliable; otherwise reflection can preserve, or even introduce, errors.

2.2 Test-Time Compute

Well-established historical inference-time strategies for improving reasoning performance include:

Best-of-N Sampling (BoN): Generate N independent random solutions and select the best using a given verifier:

$$\hat{\tau} = \arg \max_{\tau_i \sim \pi_\theta(\cdot|x),\ i \in [N]} v_\phi(x, \tau_i), \tag{1}$$

Beam Search: Maintain a beam of partial solutions, expanding promising candidates based on a given reward model's scores.

Monte Carlo Tree Search (MCTS): Explore multiple reasoning paths based on how a given verifier scores each one.

These methods established the core inference-time toolkit for verifier-guided reasoning, though later work (Section 3) identifies important limitations and proposes refinements to each.

2.3 Post-Training and Alignment

Although inference-time strategies have shown to improve the model's performance significantly, they are still not widely used in production due to their required additional compute for each prompt. Instead, these strategies often feed into an LLM's post-training process: they generate high-quality datasets for training reward models, for policy optimization using reinforcement learning, or are used directly for supervised fine-tuning of the model.

ReST (Reinforced Self Training) methods iteratively improve models using their own generated data. At each step, we sample (generate) reasoning traces using the current policy, filter high-quality traces that align with our dataset or reward model, and use them for improving the model, e.g., using supervised fine-tuning.

This iterative bootstrapping loop is loosely analogous to a common two-system view of human cognition, in which deliberate, effortful reasoning ("System 2") can, with practice, become fast and automatic ("System 1"); test-time compute plays the role of the former, while the resulting fine-tuned policy plays the role of the latter. This framing is a heuristic analogy rather than a technical claim, but it helps motivate why self-training methods can become more scalable as the number of domains and tasks grows: the model needs less explicit supervision once a behavior has been internalized.

2.4 Verification

Verification is valuable precisely because some properties of a reasoning trace can be checked more reliably than they can be generated. A verifier can therefore act as an auxiliary decision mechanism during inference and as a source of supervision during post-training. The critical question is not merely whether a score is available, but whether the score measures the property required by the downstream algorithm.

Self-consistency improves performance by sampling multiple reasoning paths and selecting the most frequent answer.

Using labeled datasets is the simplest reliable method. [3] shows that the granularity of these labels is pivotal. **Process supervision** (where each thought or reasoning step is labeled) significantly outperforms **outcome supervision**, especially as the number of candidate solutions increases. This has led researchers to pursue **reward models**, which automate this labeling process by using another model that rewards each outcome or each step.

Outcome Reward Models (ORMs) provide feedback only at the final step. Given a complete solution $s = (s_1, \ldots, s_K)$, an ORM outputs a score r_s indicating the likelihood of correctness:

$$\mathcal{L}_{\text{ORM}} = y_s \log r_s + (1 - y_s) \log(1 - r_s), \tag{2}$$

where y_s indicates whether the final answer is correct. Training an ORM requires an outcome-level-labeled dataset.

Process Reward Models (PRMs) provide step-level feedback. A PRM scores each step s_k individually:

$$\mathcal{L}_{\text{PRM}} = \sum_{k=1}^{K} y_{s_k} \log r_{s_k} + (1 - y_{s_k}) \log(1 - r_{s_k}), \tag{3}$$

where y_{s_k} indicates whether step k is correct.

Naturally, training a PRM should require a process-level-labeled dataset, but labeling each reasoning step by human annotation is a very expensive goal, especially on complex tasks that require experts. This has led researchers to pursue algorithms that use an outcome-level-labeled dataset to train a PRM. A primary approach is to use ORM datasets to make PRM estimates. In Math-Shepherd [4], the quality of a step is estimated by sampling completions and checking if they lead to correct answers. In Monte Carlo Tree Search estimation, MCTS guides exploration and provides dense value estimates. This is done either via a **Soft Estimate (SE)** or a **Hard Estimate (HE)**:

$$\text{SE}(s_k) = \frac{1}{N} \sum_{j=1}^{N} \mathbf{1}[a_j = a^*],$$

$$\text{HE}(s_k) = \mathbf{1}[\exists j : a_j = a^*],$$

where a^* is the correct answer and a_j is the answer from the j-th completion.

3 Review Methodology and Six-Category Taxonomy

The review uses one primary taxonomy, shown in Figure 1, to organize the literature according to the *main contribution* of each work. This choice reflects how the field has developed: some papers propose a new verifier or supervision mechanism, whereas others improve search, expose failure modes, establish benchmarks, or reduce cost. These contribution types should not be collapsed into a single architectural label such as "PRM." The six categories are: **(1) methodological refinements**, which improve search procedures, training objectives, or credit signals; **(2) critical analyses and corrective methods**, which identify scaling failures, reward hacking, semantic errors, or invalid assumptions; **(3) benchmarks and diagnostics**, which measure step-level error detection and verifier reliability; **(4) alternative verifier formulations**, which derive or represent verification signals in new ways; **(5) critique-based and grounded verification**, which uses natural-language reasoning, tools, or domain rules to support judgments; and **(6) efficiency-oriented methods**, which reduce annotation, training, or inference cost.

The categories are contribution-oriented rather than mutually exclusive at the mechanism level. A method may, for example, introduce an alternative verifier and also reduce cost; it is placed in the category that best captures its primary research claim, while secondary roles are discussed in the text. Section 4 later provides a functional crosswalk showing how signals are used inside reasoning systems. That crosswalk is an analytical lens within this taxonomy, not a second taxonomy or a competing partition of the literature.

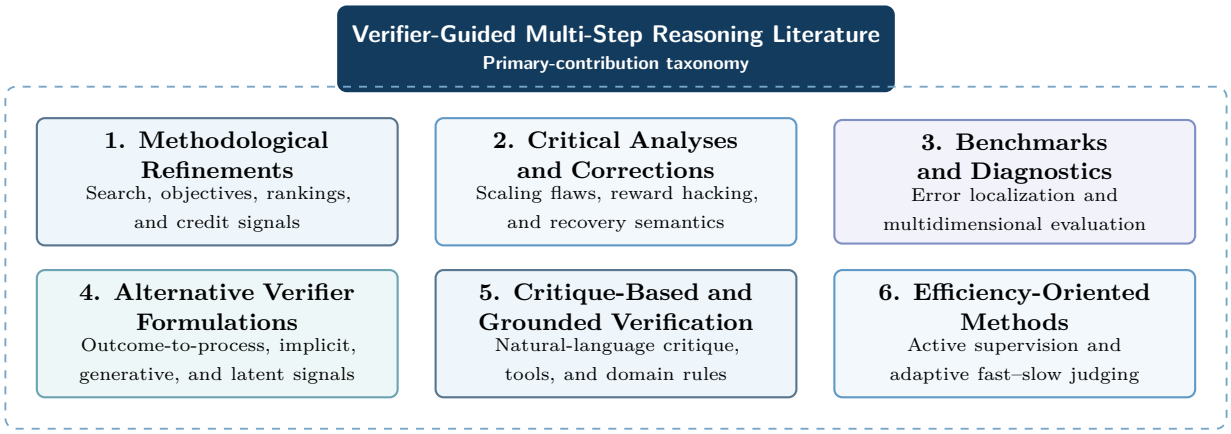

Figure 1. The single six-category taxonomy used in this survey. Papers are grouped by their primary technical contribution; overlaps in mechanism or downstream use are handled through the functional crosswalk in Table 1.

3.1 Methodological Refinements

3.1.1 Search and Tree-Construction Refinements

OmegaPRM [5] improves MCTS by doing a binary search to efficiently locate the first error in a reasoning trace. ReST-MCTS* [6] showed that the PRM can be used both in inference for guiding reasoning traces and in training for rewarding the model. This creates a positive feedback loop where the mutual training and bootstrapping of policy and value model leads to continuous improvement. [7] uses BoN or critique models in each step of MCTS trajectories. [8] propose consensus filtering, retaining only instances where MCTS estimation and critique models agree on error locations, significantly improving data efficiency and PRM quality.

3.1.2 Training Objectives and Credit Signals

The standard PRM objective is given in Eq. 3. Various methods introduce alternative objective functions to improve performance. Tree-PLV [9] uses the pairwise Bradley-Terry objective $\mathcal{L} = -\log \sigma(r(x, y^+) - r(x, y^-))$. [10] uses a Plackett-Luce ranking loss $\mathcal{L} = -\frac{1}{|C|} \sum_t \log \frac{\exp(Q_{c_t})}{\sum_{q \le t} \exp(Q_{c_q}) + \sum_w \exp(Q_w + \zeta)}$. [11] uses an MSE loss, but trains the LLM using Best-of-4 as a base model and uses advantages instead of Q-values in the beam

search and RL. rStar-Math [12] combines this technique with MCTS in two stages to train the PPM more efficiently.

3.2 Critical Analyses and Corrective Methods

MCTS criticism: [13] identify that verifier-guided search exhibits diminishing returns and can underperform simple repeated sampling at scale due to "verifier failures" (imperfect verifiers misranking and pruning valid paths, similar to reward hacking). [14] also showed both theoretically and empirically that an implicit PRM can be obtained at no additional cost (with no MCTS).

BoN criticism: [15] showed both theoretically and empirically that BoN is not scalable, as increasing N increases the chances of reward hacking, and that the best N is suboptimal unless under strong coverage assumptions and a carefully chosen N. Therefore, they propose InferenceTimePessimism as an alternative.

Credit assignment criticism: The standard RL objective for PRMs uses sum-form credit assignment, $Q^\pi(s_t, a_t) = \mathbb{E}[\sum_{i \geq t} \gamma^{i-t} r_i^p]$. [16] show this induces reward hacking, causing models to exploit high-reward steps. They instead propose min-form credit assignment,

$$G(s_t, a_t|\tau) = \min_{i \geq t} r_i^p, \tag{4}$$

which constrains the value range and significantly stabilizes training.

Error cessation criticism: [17] argues that standard PRM labels become structurally wrong for reflective long chains of thought. Existing datasets often retain only the prefix through the first error or mark every later step incorrect. The paper instead defines *error propagation*, where a step continues reasoning from an uncorrected false premise, and *error cessation*, where a step identifies the mistake or starts a valid alternative path. Their solution is a strong LLM judge that applies these context-dependent rules to segmented long-CoT traces, improving step-guided search and showing greater robustness than a Monte-Carlo-labeled counterpart.

3.3 Benchmarks and Diagnostic Evaluation

Although MCTS-based methods have seen substantial improvements over the years, they have been criticised both theoretically (as mentioned above) and empirically, with critique models increasingly outperforming them in recent benchmarks.

ProcessBench [18] introduces a new step-level-annotated dataset for Olympiad-level math problems and shows that critic models outperform MCTS-based PRMs on the new benchmark. PRMBench [19] introduces another benchmark with similar results. PRMBench uses multiple "wrong" labels along multiple dimensions such as circular logic, redundancy, and soundness.

3.4 Alternative Verifier Formulations

3.4.1 Outcome-to-Process Verification

[20] showed that outcome supervision is theoretically no more statistically difficult than process supervision and that it can be transformed into process supervision.

As a better alternative to MCTS, AutoPSV [21] proposes a rollout-free route from outcome to process supervision. Using an outcome-supervised verifier f_θ for reasoning over every prefix, it computes $\Delta_t^{\text{conf}} = (f_\theta(S^{1:t+1}; q) - f_\theta(S^{1:t}; q))/f_\theta(S^{1:t}; q)$ and applies a thresholded first-error rule to convert confidence drops into process labels. A second verifier is then trained with the automatically generated labels. [14] showed both theoretically and empirically that an implicit PRM can be obtained at no additional cost by parameterizing the outcome reward similarly to how DPO does it, i.e., $r_\phi^t = \beta \log \frac{\pi_\phi(y_t|\mathbf{y}_{<t})}{\pi_{\text{ref}}(y_t|\mathbf{y}_{<t})}$.

3.4.2 Generative, Implicit, and Latent Representations

[22] reformulate verification as next-token prediction. Instead of a separate classification head, the verifier produces a "Yes" or "No" token, directly leveraging LLM generation capabilities. GenRM further unifies generation and verification via shared training, benefiting both tasks.

[23] uses a simple centroid-based classification on a frozen model's hidden states across a reasoning trace and shows competitive results despite using a lot less compute. CLUE uses outcome labels only, but only works for a single frozen model, so it is not considered a general PRM.

3.5 Critique-Based and Grounded Verification

AutoMathCritique [24] introduces a framework for both parallel and sequential test-time scaling and post-training with a two-player paradigm separating reasoning (actor) and critique models, and shows that it raises the performance ceiling in self-training. [24] provide natural language feedback from an LLM (possibly the same model). GenPRM [25], ThinkPRM [26], and R-PRM [27] propose a separate critic model capable of CoT reasoning itself. RISE [28] tries to iteratively improve both the reward and policy model jointly.

3.5.1 Grounded and Domain-Specific Verification

GroundedPRM [29] uses math checkers like Wolfram Alpha and SymPy to improve performance on math tasks. VPRM [30] defines deterministic domain rules to improve performance on risk-of-bias assessment for medical evidence synthesis. [31] leverages multifaceted evaluation criteria to perform better on PRMBench's multi-dimensional labels.

3.6 Efficiency-Oriented Verification

Although critique models provide much higher accuracy and a higher ceiling for LLM performance after training, the cost of using another LM for each verification step is large.

Dyve [32] uses a simpler and faster model (e.g., with no CoT reasoning) for easier reasoning steps, where a shorter feedback would suffice. ActPRM [33] uses multiple binary PRM heads on a shared backbone, and only queries label supervision when there is model disagreement or low confidence.

4 Review of Key Papers through a Functional Lens

Section 3 established the survey's single six-category taxonomy and assigned papers according to their primary contribution. This section keeps that taxonomy fixed but reviews the papers through a complementary systems-level question: *what function does the verifier signal perform in the reasoning pipeline?* The resulting view is useful because the same technical method can generate supervision, guide search, support refinement, train a policy, or diagnose errors, and these uses impose different requirements on calibration, semantics, latency, and robustness.

Table 1 is therefore a *functional crosswalk*, not another taxonomy. Its rows are deliberately non-exclusive: a paper may appear in more than one functional discussion, and papers that share a row may belong to different categories in Figure 1. The table links the contribution-oriented organization of Section 3 to the system roles emphasized in the paper-by-paper review that follows.

[3] is the foundational paper for modern process supervision in mathematical reasoning. The paper introduced PRM800K, a large dataset of human-labeled solution steps for MATH problems, and compared outcome-supervised and process-supervised verifiers. Its core empirical result is that PRMs outperform ORMs when used to select from many sampled solutions, and that the gap grows with the number of candidates. The paper also motivates active learning for process annotation, because most randomly sampled steps are uninformative. Its lasting contribution is not only a dataset but also a shift in the evaluation target: a good reasoning model should not merely reach the right answer, but should take a valid path to it.

Functional role of the signal	Representative papers	System-level contribution	Main limitation
Reference supervision and diagnostics	[3], [18]	Establishes step-level correctness and first-error localization as stronger diagnostic targets than outcome accuracy alone.	Expensive labels and limited domain coverage.
Scalable label construction	[4], [5], [21], [33]	Converts outcomes, confidence changes, or selective judgments into process supervision.	Proxy errors and model uncertainty can bias the labels or selected data.
Search and choice guidance	[6], [9], [10], [12]	Ranks steps, estimates values, or guides tree expansion and candidate selection.	Search quality depends strongly on verifier calibration and coverage.
Reasoning-intensive judgment and refinement	[22], [25], [26], [27], [32]	Treats verification as a reasoning task and can adapt judgment depth to apparent difficulty.	Higher latency and vulnerability to unfaithful critiques or routing errors.
External or internal evidence checking	[29], [30], [23], [17]	Grounds judgments in tools, rules, hidden states, or transition-aware recovery evidence.	Requires parsable steps, white-box access, strong judges, or domain-matched evidence.
Training-time feedback and self-improvement	[24], [28]	Uses critique or self-verification to construct training data or optimize the reasoning policy.	Unstable or exploitable rewards can amplify verifier blind spots.
Reliability auditing	[8], [19], [13], [15]	Reveals gaps among answer selection, step-error detection, search scaling, and reward robustness.	Existing evidence remains math-heavy and incomplete for interactive or multimodal reasoning.

Table 1. Functional crosswalk of the reviewed literature. Unlike the six-category taxonomy in Figure 1, this table does not partition papers; it shows the role played by a verifier signal within the reasoning system.

Math-Shepherd addresses the scalability problem left by PRM800K [4]. Instead of relying on human step labels, it estimates step quality by sampling completions from a partial reasoning prefix and checking whether the resulting final answer is correct. The paper proposes soft and hard estimates, where the soft estimate counts the fraction of successful rollouts and the hard estimate asks whether any rollout succeeds. In notation, a typical automated target can be written as

$$\hat{q}(s_t, a_t) = \frac{1}{M} \sum_{m=1}^{M} \mathbb{I}\left[\mathrm{Ans}\big(\tau_{t:T}^{(m)}\big) = y^*\right], \tag{5}$$

where completions are sampled from the prefix $(x, a_{\le t})$. Math-Shepherd shows that automated process supervision can improve both verification and reinforcement of LLM reasoning. The limitation is that the target in Eq. 5 is not pure local correctness; it is a recoverability estimate under a particular completion policy.

ReST-MCTS* combines self-training with PRM-guided tree search [6]. In each iteration, MCTS explores reasoning trajectories using PRM values, selected high-quality traces are used for supervised fine-tuning, and search statistics provide updated process signals for the verifier. This creates a mutual bootstrapping loop between policy and PRM. The paper is important because it treats process supervision as an online data-generation mechanism rather than a static annotation resource. It also illustrates a recurring pattern in this literature: search improves the data used for training, and the improved model changes the distribution on which the verifier is evaluated.

Rewarding Progress proposes automated process verifiers that reward progress rather than absolute correctness [11]. The key idea is to compare the future solvability of a state after a candidate step with the solvability of the state before that step, often using a complementary prover to estimate whether a prefix has moved the solution closer to success. This provides a natural advantage-like interpretation of process reward. Rewarding Progress is also notable for scaling automated process verifiers and showing that process signals can be derived without manually labeling every step. Its limitation is that the reward is defined relative to a complementary prover and therefore depends on the capabilities and biases of that prover.

[34] analyze how to allocate test-time compute among strategies such as best-of-N, beam search, and verifier-guided search. Their results show that inference-time computation can sometimes be more effective than increasing model size, but that the optimal strategy depends on task difficulty and the quality of the verifier. This paper provides an important systems-level lens for the field: verifiers are not only training objects but also computational resources. The best use of a fixed budget may be to sample more answers, search deeper, or evaluate more carefully depending on the problem.

rStar-Math studies whether small LLMs can reach strong mathematical reasoning through self-evolved deep thinking [12]. The system uses MCTS-style exploration and process supervision to generate higher-quality reasoning data, then iteratively improves the policy. Its importance lies in demonstrating that verifier-guided test-time search and self-training can partially compensate for smaller model size. It also reinforces the connection between process verification and planning: a verifier can be used to evaluate partial branches, not only completed answers.

ProcessBench shifts attention from improving generators to evaluating verifiers [18]. It asks models to identify the first erroneous step in mathematical reasoning traces, or return a special value when no error is present. This is stricter than final-answer accuracy because a model must localize the causal process error. ProcessBench reveals that many PRMs trained for reranking do not necessarily generalize to hard first-error detection, while prompted LLMs can sometimes provide useful critique. The benchmark therefore separates process understanding from answer-level selection.

Tree-PLV advances process verification through tree-based preference learning [9]. Rather than assigning independent correctness labels to steps, it builds reasoning trees and learns pairwise preferences among sibling steps or paths. This formulation matches how verifiers are used in search: among several possible next steps, the system needs to choose the most promising one. Tree-PLV shows that preference supervision can be more suitable than scalar classification when the downstream use is selection under a branching reasoning process.

Critique-model supervision, studied by [24], reframes verification as natural-language feedback in a two-player actor-critic setting. A critique model can judge a response, explain errors, and support both test-time revision and training-time data construction. This line of work is important because it connects PRMs with a broader literature on self-correction and feedback. It also highlights a failure of naive self-reflection: a model may revise correct reasoning into incorrect reasoning unless feedback is accurate and targeted.

PQM, or Process Reward Model with Q-value Rankings, provides a value-based interpretation of step rewards [10]. Instead of asking whether a step is locally correct, it asks whether taking that step increases the probability of eventual success. PQM trains with ranking constraints between steps with different Q-values, which can be expressed abstractly by a pairwise loss

$$\mathcal{L}_{\text{rank}} = \sum_{(i,j)} \max\left(0,\ m - \operatorname{sgn}(q_i - q_j)\big(f_\phi(s_i, a_i) - f_\phi(s_j, a_j)\big)\right), \tag{6}$$

where m is a margin. The paper clarifies why absolute step labels may be insufficient: a step can be correct but low-value if it leads to a dead end, or imperfect but high-value if it preserves recoverability.

[20] challenge the assumption that process supervision is fundamentally more sample-efficient than outcome supervision. From a theoretical perspective, they show that outcome supervision can be transformed into process supervision under suitable assumptions and that the statistical separation between the two is not as simple as early empirical results suggest. This does not invalidate PRMs; rather, it implies that their value may come from algorithmic alignment with search, interpretability, or data efficiency under practical constraints, not from an unconditional information-theoretic advantage.

GenPRM scales test-time compute on the verifier side [25]. Instead of treating a PRM as a one-shot scalar classifier, it prompts or trains a verifier to reason generatively about the correctness of each step and then aggregates multiple verifier judgments. This method is especially relevant for hard mathematical problems where detecting a step error requires non-trivial derivation. Its main trade-off is cost: spending more compute on the verifier can improve accuracy, but it competes with spending compute on generating additional candidate solutions.

OmegaPRM improves mathematical reasoning by automated process supervision [5]. It recognizes that checking every step with many rollouts is expensive and proposes a more efficient divide-and-conquer procedure for locating the first error in a reasoning trace. By constructing large-scale process labels from final-answer checks, OmegaPRM bridges the gap between human-labeled PRMs and fully outcome-supervised training. Its broader contribution is to make automated process labeling a computational problem: the objective is not only to get labels but to get them at an acceptable cost.

AutoPSV proposes a rollout-free route from outcome to process supervision [21]. It first trains an outcome-supervised verifier f_θ to estimate final-answer correctness from every prefix. For two consecutive prefixes, it computes

$$\Delta_t^{\text{conf}} = \frac{f_\theta(S^{1:t+1}; q) - f_\theta(S^{1:t}; q)}{f_\theta(S^{1:t}; q)}, \tag{7}$$

and applies a thresholded first-error rule to convert confidence drops into process labels. A second verifier is then trained with the automatically generated labels. Across GSM8K, MATH, HellaSwag, Winogrande, and ANLI, the process-enhanced verifier usually improves or matches outcome supervision and self-consistency, although not in every model–dataset combination. Its main efficiency result is especially notable: on the authors' cost accounting, confidence-based labeling processes roughly one pass over a trace, whereas MCTS labeling requires many continuation tokens per step. AutoPSV can also label generated questions without gold answers once the outcome verifier has been trained. The limitation is semantic: a confidence change is evidence about the outcome verifier's belief, not an independently verified proof that the new step is correct.

ActPRM reframes scalable process supervision as an active-learning problem [33]. A shared LLM backbone is equipped with multiple independently initialized classification heads. Their mean confidence estimates aleatoric ambiguity, while their standard deviation estimates epistemic disagreement. A trajectory is sent to the expensive QwQ-32B judge only when a step at or before the predicted first error is uncertain by either criterion. In pool-based experiments, ActPRM matches an average ProcessBench F1 of 0.673 with half the annotation budget and reaches 0.680 with 62.5% of the budget. In the one-shot scale-up, filtering 1,061,763 NuminaMath trajectories retains 563,030 for labeling, after which ActPRM reaches 0.750 average F1 on ProcessBench; ActPRM-X, which continues training a stronger existing PRM on the selected data, reaches 0.760 on ProcessBench and 0.667 on PRMBench. These results show that label selection can matter as much as label volume. However, the method inherits the standard first-error truncation assumption and relies on ensemble uncertainty and a strong neural annotator rather than independently verifiable step labels.

Generative Verifiers formulate reward modeling as next-token prediction [22]. Rather than adding a separate scalar head, the model generates verification tokens such as affirmative or negative judgments. This brings reward modeling closer to the pretrained language-model objective and allows shared generative capacity to support both answering and judging. The paper is part of a broader shift from discriminative verifiers to generative verifiers, in which the verifier's latent reasoning ability becomes a resource.

CLUE, introduced in *Your Reasoning Model is Secretly a Reward Model*, asks whether a frozen reasoning model already contains a correctness signal in its hidden-state trajectory [23]. For every labeled solution T, it extracts hidden states at the beginning and end of the explicit thinking span and computes the all-layer activation delta

$$\Delta h(T) = h_{\text{end}}(T) - h_{\text{start}}(T). \tag{8}$$

Correct and incorrect experience sets are each compressed into a centroid matrix. A new trajectory is classified by its layer-averaged Euclidean distance to the two centroids and reranked by distance to the success centroid. No gradient update is performed. On AIME 2024/2025, CLUE substantially outperforms optimization-free textual self-judgment, while frontier full-trace judges remain stronger overall. For Nemotron-1.5B on AIME 2024, top-majority@16 improves majority@64 from 56.7% to 70.0%. The paper also reports gains on GPQA and WebInstruct-verified, with especially clear benefits for smaller or less-calibrated reasoners. Its limitations delineate a new verifier category rather than a universal replacement: it requires white-box hidden-state access, reliable binary experience labels, reasonably stable prompt formatting, and domain-aligned centroids; performance drops when experience is too small or transferred across mismatched domains.

Free Process Rewards without Process Labels further blurs the distinction between outcome and process supervision [14]. The paper shows that implicit process rewards can be derived without explicit process labels, using an outcome-reward parameterization related to preference optimization. This result is important because it suggests that some process information is already latent in outcome-supervised or preference-trained models. However, such implicit rewards still need careful evaluation: a free signal may be useful for reranking but not necessarily safe for RL or first-error localization.

The Lessons of Developing Process Reward Models in Mathematical Reasoning provides a practical audit of PRM design and evaluation [8]. The paper emphasizes that PRM quality cannot be judged solely by best-of-N answer accuracy, because a verifier may select correct answers for the wrong process reasons. It argues for combining response-level and step-level evaluations and studies how data quality, annotation schemes, and evaluation protocols influence conclusions. Its role in this survey is diagnostic: it cautions against overinterpreting any single benchmark number as evidence of process understanding.

ThinkPRM, or Process Reward Models That Think, shows that long-CoT reasoning models can become strong process verifiers with modest process supervision [26]. Instead of outputting a scalar score immediately, the verifier reasons about the step before judging it. This is especially effective on difficult or out-of-domain traces where shallow classifiers fail. ThinkPRM therefore supports the hypothesis that verification is itself a reasoning task. The remaining challenge is to manage the additional token cost and to ensure that the generated rationale is faithful to the score.

Beyond the First Error argues that standard PRM labels become structurally wrong for reflective long chains of thought [17]. Existing datasets often retain only the prefix through the first error or mark every later step incorrect. The paper instead defines *error propagation*, where a step continues reasoning from an uncorrected false premise, and *error cessation*, where a step identifies the mistake or starts a valid alternative path. A strong LLM judge applies these context-dependent rules to segmented long-CoT traces, producing 1.7 million step labels for a 7B PRM. The resulting model reaches 0.816 PRM@64 on MATH500 and 0.828 step-level F1 in the authors' evaluation, while also improving step-guided search and showing greater robustness than a Monte-Carlo-labeled counterpart. The paper's diagnostic is as important as its model: established open PRMs lose 10 to more than 50 percentage points when moving from error-free correct solutions to correct solutions that contain and recover from an intermediate error. The method nevertheless depends on a capable LLM annotator, focuses on mathematical long-CoT data, and leaves broader scaling evidence open.

Stop Summation analyzes a failure mode of PRM-based reinforcement learning [16]. If a policy is rewarded by summing process scores, it may learn trajectories with many locally high-scoring steps even if a later step invalidates the solution. The min-form objective in Eq. 4 instead makes the weakest future step dominate the return, which better matches the logic of mathematical correctness. This paper is important because it shows that a PRM useful for inference-time selection can become dangerous when used naively as a dense RL reward.

Scaling Flaws of Verifier-Guided Search studies what happens as verifier-guided search budgets increase [13]. The paper shows that deeper or broader search does not guarantee improvement over repeated sampling. Imperfect verifiers can prune correct branches early or over-score invalid branches, making search quality non-monotonic. This result complements the compute-optimal view of [34]: scaling test-time compute is valuable only when the allocation algorithm and verifier reliability are matched.

Step-level Verifier-guided Hybrid Test-Time Scaling extends this discussion by proposing a training-free hybrid inference paradigm [7]. The paper first introduces conditional step-level self-refinement, where a step is revised only when a PRM indicates that refinement is needed and the revised step improves the score. It then combines this sequential refinement with parallel scaling methods such as best-of-N and MCTS at the step level. Experiments on MATH500, AIME24, and GPQA Diamond across instruction-tuned models from 3B to 14B show that fine-grained hybrid scaling can substantially improve reasoning without additional training. The method is important because it treats parallel and sequential test-time scaling as complementary rather than competing. Its limitation is that it relies on a strong step-level verifier and still increases inference cost when multiple refinement rounds are used.

DG-PRM, or Dynamic and Generalizable Process Reward Modeling, focuses on generalization beyond fixed mathematical correctness criteria [31]. The method constructs a reward tree from LLM judgments, stores multifaceted evaluation criteria, dynamically selects criteria relevant to each step, and uses Pareto dominance estimation to form positive and negative pairs from multi-dimensional reward signals. This design addresses two weaknesses of many PRMs: fixed evaluation criteria and uniform negative rewards. By showing gains on PRMBench and broader preference-style benchmarks, DG-PRM suggests that future process verifiers may need explicit criterion selection, especially for domains where errors differ in severity or type.

R-PRM proposes reasoning-driven process reward modeling [27]. Existing PRMs often output scores directly, while R-PRM uses stronger LLMs to generate seed data that teaches the model a more comprehensive step-by-step evaluation procedure. It then applies preference optimization without requiring extra annotated data and uses inference-time scaling to further improve judgments. On ProcessBench and PRMBench, R-PRM reports large F1 improvements over strong baselines. The contribution is conceptually close to ThinkPRM and GenPRM: a verifier should not merely recognize surface patterns of correctness but should actively reason through the step before scoring it.

PRMBench provides a fine-grained benchmark for process-level reward models [19]. It contains 6,216 problems and 83,456 step-level labels, and evaluates PRMs along multiple dimensions such as simplicity, soundness, and sensitivity. The paper evaluates both open-source PRMs and LLMs prompted as critics, revealing that many current PRMs struggle with false positives and that best-of-N evaluation can be weakly correlated with true process-level error detection. PRMBench is especially important because it expands the benchmark view from first-error localization alone to a richer set of process-quality dimensions.

Trust, But Verify introduces RISE, an online reinforcement learning framework that trains problem solving and self-verification together using verifiable rewards [28]. In each iteration, the model generates solutions, the outcome verifier assigns rewards, and the same problem-solution-reward triples are transformed into verification prompts. The policy is then optimized jointly on generation trajectories and verification trajectories. In simplified form, RISE trains on a mixed batch

$$\mathcal{B} = \mathcal{B}_{\text{solve}} \cup \mathcal{B}_{\text{verify}}, \tag{9}$$

where the verification target is derived from the same verifiable reward used for solution correctness. This directly addresses superficial self-reflection: the model learns not only to produce answers but also to judge its own on-policy outputs. The main limitation is that the approach requires reliable verifiable rewards, which are easiest to obtain in mathematics and code.

GroundedPRM combines structured search with execution-grounded step supervision [29]. MCTS constructs diverse partial paths; during simulation, a mathematical tool such as Wolfram Alpha checks intermediate operations, and the final answer is compared with ground truth. For a rollout from step i, the framework aggregates discounted step labels $v_j \in \{-1, 1\}$ and final correctness $F \in \{-1, 1\}$:

$$u_i = \frac{1}{T-1-i} \sum_{j=i+1}^{T-1} d_j v_j + \beta F. \tag{10}$$

The reward is backpropagated through the tree, and 40K retained traces are formatted as generative examples containing a binary judgment and tool-derived rationale. GroundedPRM obtains 39.7 average F1 on ProcessBench, a reported 26% relative improvement over the strongest automatically labeled baseline in its comparison while using about one tenth as many samples. In reward-guided greedy search with eight candidate next steps, it reaches 42.4% average accuracy across six math benchmarks, exceeding the compared human-, mixed-, and automatically supervised PRMs. Ablations are instructive: outcome-only supervision collapses because of credit misattribution, while tool-only supervision misses deeper logical errors; their combination and rationale generation are both important. The framework is therefore grounded rather than fully automatic in a domain-independent sense: it requires steps that can be translated into queries accepted by a suitable external verifier.

Beyond Outcome Verification introduces Verifiable Process Reward Models (VPRMs) for structured reasoning tasks whose intermediate decision path is defined by deterministic domain rules [30]. In risk-of-bias

assessment for medical evidence synthesis, every generated step contains a step identifier and a discrete label. Rule-based functions check whether both agree with guideline-derived targets, and their rewards are combined with a verifiable final risk label:

$$R_{\text{VPRM}} = \lambda_{\text{proc}} \sum_{k=1}^{K} \mathbf{1}[\hat{\ell}_k = \ell_k^*] + \lambda_{\text{out}} \, \mathbf{1}[\hat{y} = y^*], \tag{11}$$

where $\hat{\ell}_k$ is the predicted label for step k, ℓ_k^* is the guideline-derived target label, $\hat{y}$ is the final predicted risk assessment, and y^* is the ground-truth outcome. The authors train Qwen2.5-7B with GRPO and DAPO and show that process-verifiable RL consistently improves over outcome-only RL. The GRPO-VPRM model reaches 76.7 macro-F1 on Cochrane Forest, compared with 70.2 for verifiable outcome reward alone, and raises reasoning–decision coherence to 89.5%. It also outperforms neural-judge process rewards in the controlled comparison. VPRM is an important non-mathematical demonstration that process rewards themselves can be deterministic, transparent, and useful for RL. Its scope is necessarily narrower than a general PRM: it requires complete domain rules, a verifier-compatible output format, and guidelines whose omissions or biases do not invalidate the reward.

Finally, Is Best-of-N the Best of Them? provides a theoretical account of coverage, scaling, and optimality in inference-time alignment [15]. The paper shows that best-of-N can be optimal under strong coverage assumptions and a carefully chosen N, but that increasing N indefinitely can produce reward hacking when the reward model is imperfect. Its InferenceTimePessimism algorithm uses a more conservative uncertainty-aware selection procedure and is proved to be scaling-monotonic. This result is highly relevant to verifier-guided reasoning because many PRM systems implicitly assume that more candidates or more search will help. The theory shows that compute must be paired with pessimism or uncertainty control when verifier errors are possible.

5 Results: Comparative Synthesis Across Studies

While Section 3 organizes the literature by primary contribution, this section synthesizes empirical evidence from the subset of studies that share enough experimental structure—including generator model, benchmark, candidate-set size N, and selection rule—to support a meaningful comparison. Direct comparisons are limited to matched settings reported within a paper or to shared candidate pools and protocols; results from unmatched settings are retained only as contextual evidence and are labeled accordingly. Throughout, we distinguish three notions of efficiency that are frequently conflated in this literature: *inference efficiency* (compute needed to solve test questions), *training efficiency* (compute or data needed to train the verifier or policy), and *sample efficiency* (number of candidate solutions or RL iterations needed to reach a target accuracy).

5.1 Matched Accuracy Comparisons

Table 2 summarizes comparisons in which the candidate solutions, generator model, and benchmark are shared across the compared methods, making the accuracy gap directly attributable to the verifier.

A useful complementary view is which methods *dominate* an alternative, in the sense of improving accuracy without adding meaningful inference cost: PQM dominates BCE-based PRMs in its controlled experiments, since it changes only the training loss while keeping the same verifier architecture at inference time. Tree-PLV dominates Math-Shepherd under the matched Mistral experiments while reportedly requiring only 22.7% as much training data. OmegaPRM dominates earlier annotation datasets because inference remains a conventional PRM forward pass while data generation is far more efficient. Qwen2.5-Math-PRM-7B dominates its PRM800K and Math-Shepherd counterparts in both BoN and ProcessBench evaluations. Finally, PURE's min-form credit assignment (Eq. 4) dominates sum-form assignment because sum-form training becomes unstable while min-form training remains stable.

Matched setting	Best method	Comparable results	Main conclusion
GPT-4 generator, MATH500, BoN@1860	Human PRM	PRM 78.2, ORM 72.4, majority 69.6	Process supervision gives +5.8 points over ORM.
LLaMA2-70B MetaMATH, 256 candidates	Math-Shepherd	GSM8K 93.2 vs. ORM 91.8; MATH500 44.5 vs. 40.4	Automatic PRM helps much more on the harder benchmark.
Mistral-7B, BoN@64	Tree-PLV	GSM8K 82.79 vs. Math-Shepherd 74.91; MATH500 26.8 vs. 21.2	Clear improvement with the same generator.
MetaMATH-Mistral-7B, BoN@64	Tree-PLV	GSM8K 87.72 vs. Math-Shepherd 87.11; MATH500 37.2 vs. 35.4	Smaller but consistent gain with a stronger generator.
MetaMATH-Mistral-7B, BoN@128	PQM	MATH500 44.6, BCE 42.0, ORM 38.2	Ranking loss beats classification at no extra inference cost.
Same setting, GSM-Plus	PQM	65.2–66.0 vs. BCE 61.72, ORM 58.33	About +4 points over BCE and +7 over ORM.
Gemma2-27B, weighted vote@64	OmegaPRM	MATH500 58.2 vs. Math-Shepherd 57.4, PRM800K 57.2; GSM8K 92.2 vs. PRM800K 91.7	Best annotation method, though gains are modest.
Qwen2.5-Math-7B generator, BoN@8	Qwen2.5-Math-PRM-7B	Seven-task average 67.6 vs. majority 66.2, PRM800K 64.9, Math-Shepherd 64.3	Best 7B discriminative PRM in this comparison.
Three shared MATH500 candidate pools	Implicit PRM (DPO)	Average 50.4 vs. Math-Shepherd 47.8, AutoPSV 45.7	Better accuracy at much lower development cost.
Same 7B model/data, ProcessBench	GenPRM-7B	Pass@1 75.2 vs. direct generative PRM 60.0	Generative reasoning adds +15.2 F1.
R1-Qwen-14B, hardest ProcessBench subsets	ThinkPRM	Average 86.5 vs. untrained LLM-as-judge 70.3	Fine-tuning on 1K verification chains adds 16.2 F1.
Llama3-8B actor, GSM8K/MATH	Test-time critic	Accuracy 54.8/17.2 → 63.3/24.3	Critique and refinement improve a plain SFT actor.
Qwen2.5-Math-7B policy training	PURE-PRM+VR	Five-task average 53.3 vs. PRM-only 49.3, verifiable-reward-only 48.3	Dense PRM plus a small ground-truth signal is best.

Table 2. Matched-setting accuracy comparisons across reviewed papers. "Matched" indicates that generator, benchmark, and candidate-set size are held fixed across the compared rows.

5.2 ProcessBench as a Shared Evaluation Anchor

ProcessBench [18] provides one of the strongest shared evaluation anchors in this literature because it evaluates error detection directly rather than only final-answer selection. Table 3 collects full-benchmark average F1 scores reported by the reviewed studies. This is a descriptive comparison across studies, not a fully controlled ranking: model size, verifier sampling, prompting, and implementation details remain different unless explicitly matched.

Three points follow from Table 3. First, the highest overall score belongs to o1-mini, but because it is a proprietary model with an undisclosed sampling budget, it should be read as a reference point rather than a controlled comparison. Second, among open methods, GenPRM-32B with majority-of-8 sampling is the strongest, and GenPRM-7B with the same aggregation is the strongest 7B-scale generative method at 80.5. Third, Qwen2.5-Math-PRM-7B is the strongest fast (single-pass) 7B classifier at 73.5, making it the natural reference point for latency-sensitive deployment.

Figure 2 visualizes these scores, grouping methods by whether they require a single forward pass or repeated sampling with majority aggregation.

5.3 Inference Speed and Compute Trade-offs

Exact wall-clock latency is rarely reported in a directly comparable form across papers, so a fully numerical speed leaderboard is not possible; however, the qualitative computational ordering across method families is clear and is summarized in Table 4.

One of the few papers to report concrete GPU-time measurements is the implicit-PRM work [14], reproduced in Table 5.

Model	MATH (Pass@1)	AIME 2024 (%)	ProcessBench (Avg F1)
Critique Models			
GenPRM-7B	90.0	53.3	80.5
ThinkPRM-14B	∼90	∼56.7	87.3/85.7
R-PRM-7B	80.0	16.7	70.4
Novel Verifiers			
Free PRM-8B	54.4	–	–
GenRM-9B	44.6	–	–
CLUE-1.5B	–	70.0	–
AutoPSV-7B	16.9	–	–
Methodological Refinements			
rStar-Math-7B	90.0	53.3	–
ReST-MCTS*-7B	48.5	–	–
PAV-9B	∼48	–	–
PQM-7B	45.4	–	–
Tree-PLV-7B	26.8	–	–
Common Baselines			
Qwen2.5-Math-PRM-72B	–	–	78.3
Qwen2.5-Math-PRM-7B	–	–	73.5
Math-Shepherd-7B	38.0	–	31.5
GPT-4o	76.6	9.3	61.9
o1-preview	85.5	44.6	–
o1-mini	90.0	56.7	–

Table 3. Overall comparison of process reward models and reasoning methods. Bold = highest score in each column among open models. ThinkPRM's ProcessBench scores are on the two hardest subsets only.

5.4 Search Strategy versus Compute Budget

A recurring theme in the reviewed literature is that the best search strategy depends on the available test-time compute budget rather than being universally fixed.

At **small to moderate compute budgets**, verifier-guided beam search tends to dominate. The compute-optimal test-time scaling analysis of [34] shows beam search reaching approximately 27–34% on MATH at budgets where Best-of-N obtains roughly 21–29%. Process Advantage Verifiers (PAV) obtain 8–10 absolute points higher accuracy than ORM-guided Best-of-N [11], and need 1.5–5× less test-time compute to match ORM Best-of-N performance. At small budgets, PAV-guided beam search is therefore the strongest result among the reviewed papers.

At **large compute budgets**, the ordering reverses and repeated sampling becomes the safer choice. In the compute-optimal analysis, beam search initially leads but eventually falls below Best-of-N as the budget grows [34]. The Scaling Flaws analysis finds that verifier-guided search can move from more than 20% above repeated sampling at very small budgets to roughly 5% below repeated sampling on GSM8K, nearly 30% below it on MATH, and even further behind under difficult or out-of-distribution conditions [13]. The underlying cause is verifier failure: an imperfect verifier incorrectly prunes valid reasoning paths as the search tree grows.

Figure 3 sketches this qualitative crossover pattern.

The strongest overall search policy suggested by this comparison is therefore *adaptive*: use sequential revision or modest sampling on easy questions, verifier-guided beam search on medium or hard questions, and Best-of-N or repeated sampling at very high budgets or when the verifier is unreliable. [34] report that compute-

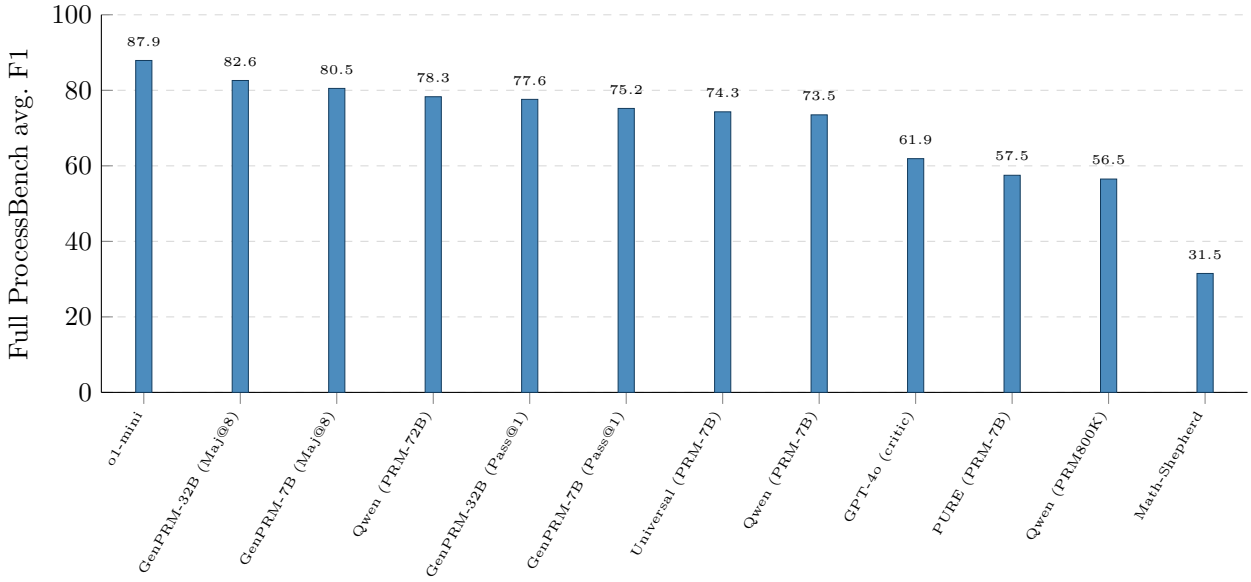

Figure 2. Full ProcessBench average F1 by method, ordered from highest to lowest.

Method family	Relative inference cost	Accuracy tendency
Majority voting	Lowest: no verifier	Strong on easy problems, limited error detection.
ORM / discriminative PRM	One forward pass per solution	Best general speed–accuracy trade-off.
PQM, Tree-PLV, Qwen PRM	Same order as ordinary PRM	Better accuracy from training changes, not extra inference cost.
Implicit PRM	Policy plus reference model	Higher than ordinary PRM unless the reference is removed.
PAV beam search	Repeated step-level scoring, sequential search	Very compute-efficient at reaching a target accuracy.
GenRM-CoT	Generates verification reasoning	Slower, usually more accurate.
ThinkPRM	Roughly 1K–5K verification tokens	Very accurate but expensive.
GenPRM	Roughly 2.8K–4.9K generated tokens per response	Strongest full-benchmark open results.
Maj@4 / Maj@8 generative PRM	4–8 generated verification traces	Highest accuracy, highest verification cost.
Critique + refinement	Actor → critic → actor refinement	Expensive, especially across multiple turns.

Table 4. Qualitative inference-cost ordering across verifier families, alongside the corresponding accuracy tendency reported in the source papers.

optimal allocation can match or exceed Best-of-256 with approximately 64 samples, a roughly 4× sample-efficiency gain, although this result does not fully account for the cost of estimating question difficulty in the first place.

5.5 Training and Data-Construction Efficiency

Table 6 summarizes reported training- and data-efficiency results, which are typically not directly comparable to one another in absolute terms but each demonstrate a large relative improvement within their own matched setting.

PAV itself costs roughly 10× more to train than an ORM, since it requires labels for many prefixes rather than only complete solutions; the originating paper argues this cost is amortized over repeated deployment

Generator	Normal PRM cost	Implicit PRM cost	Slowdown
Mistral-7B	200.9	301.6	1.50×
Llama-3.1-8B	171.1	241.7	1.41×
Llama-3.1-70B	111.1	122.2	1.10×

Table 5. Reported GPU-time cost (arbitrary consistent units, as given in [14]) for standard versus implicit PRM scoring. The relative overhead of the reference model shrinks as the generator grows larger, and the paper notes the reference can sometimes be removed with little accuracy loss.

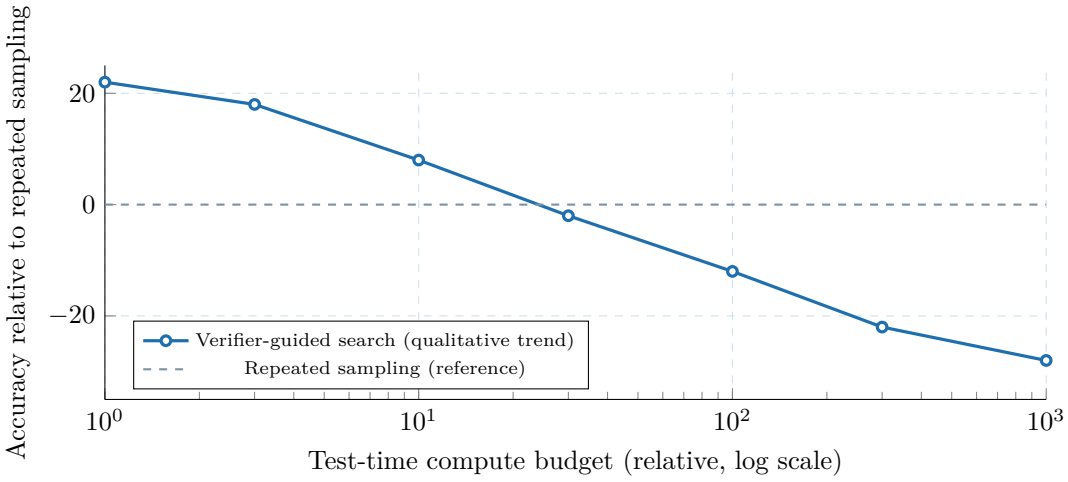

Figure 3. Qualitative illustration of the crossover reported by [13] and [34]: verifier-guided search substantially outperforms repeated sampling at small budgets, but its relative advantage shrinks and eventually turns negative as the budget grows and verifier errors accumulate. Values are illustrative of the reported qualitative trend rather than a single numerical result from one paper.

[11]. For Gemma-2B online RL, the paper reports a total compute of approximately 2.5×10^{18} FLOPs for PAV-RL versus approximately 5.9×10^{18} FLOPs for ORM-RL to reach matching accuracy roughly a 2.4× reduction in total training compute.

5.6 Best Method by Practical Objective

Table 7 consolidates the comparisons above into a practical recommendation table, organized by deployment objective rather than by paper.

Taken together, if a single method had to be chosen for each operating regime: a real, latency-sensitive system should use Qwen2.5-Math-PRM-7B or a PQM-style classifier PRM; maximum verification accuracy calls for GenPRM or ThinkPRM with verifier-side compute scaling; limited test-time compute favors PAV-guided beam search; large test-time budgets favor adaptive search or plain Best-of-N, rather than blindly continuing to scale beam search; inexpensive PRM development favors the implicit PRM; and training the reasoning policy itself is best served by combining dense process rewards with a small amount of verifiable outcome reward, as in PURE-PRM+VR.

The most important methodological lesson from this comparative synthesis is that high Best-of-N accuracy does not necessarily imply good process verification. [8] show that Monte-Carlo-labeled models can appear strong under Best-of-N evaluation while performing poorly on ProcessBench's first-error localization task. A reliable comparison between verifiers should therefore report both answer-selection accuracy and step-level error-detection accuracy rather than either alone.

6 Comparison and Discussion

Goal	Best result	Evidence
Automatic step-label generation	OmegaPRM	15M labels vs. 200K for brute-force MC at the same budget (75×).
Lowest PRM development compute	Implicit PRM (CE)	38.6–38.8× cheaper than Math-Shepherd.
Preference-based implicit PRM	Implicit PRM (DPO)	21.3–146.5× cheaper than Math-Shepherd, depending on responses per instruction.
Data-efficient generative verifier	ThinkPRM	1K reasoning chains (∼8K step labels) vs. 712K labels for a discriminative PRM.
Active human-label selection	PRM800K	Active learning is roughly 2.6× more data-efficient.
RL sample efficiency	PAV-RL	Matches ORM-RL accuracy in about 1/6 the RL iterations.
Dense-reward policy training	PURE	Needs about 30% as many steps as sparse verifiable-reward training.

Table 6. Training- and data-efficiency results reported across the reviewed papers. Comparisons are within each paper's own matched setting rather than across rows.

Objective	Recommended method
Lowest test-time overhead	Majority voting
Best fast verifier	Qwen2.5-Math-PRM-7B
Better training objective, unchanged runtime	PQM
Best preference-based classifier verifier	Tree-PLV
Best automatic annotation efficiency	OmegaPRM
Lowest verifier development cost	Implicit PRM
Best full-ProcessBench open-source accuracy	GenPRM-32B (Maj@8)
Best 7B-scale ProcessBench accuracy	GenPRM-7B (Maj@8)
Best accuracy/latency compromise at 7B scale	Qwen2.5-Math-PRM-7B
Best low-budget test-time search	PAV-guided beam search
Best high-budget search	Adaptive compute-optimal selection, otherwise Best-of-N
Best RL final performance	PURE-PRM+VR
Best RL sample efficiency	PAV-RL
Best difficult / out-of-distribution verification	Generative PRMs (e.g., ThinkPRM, GenPRM)

Table 7. Recommended method by practical deployment objective, synthesized from the matched comparisons in Sections 5 and the associated subsections.

The papers reviewed above show a clear evolution in the field. Early work established that process supervision can improve answer selection, especially in mathematical reasoning. Later work made process supervision scalable through automated rollouts, tree search, preference learning, and outcome-to-process transformations. The newest papers broaden both the evidence and the object of verification: confidence trajectories and active uncertainty reduce labeling cost; hidden-state geometry exposes internal correctness signals; external tools and deterministic rules make selected process rewards verifiable; fast–slow models adapt judgment cost; and reflective annotations continue beyond the first error. The resulting question is no longer simply whether process supervision helps, but what evidence justifies each reward, which reasoning transitions the reward represents, and what failures arise when it is optimized or scaled. The empirical comparison in Section 5 reinforces this shift: no single verifier family dominates across every axis, and the right choice depends on the compute budget, the required latency, and whether the downstream use is reranking, search, or reinforcement learning.

Design tension	Benefit	Failure mode	Representative papers
Process versus outcome labels	Step-level feedback enables localization and pruning.	Human labels are costly; automated labels can be policy-dependent.	[3], [4], [20], [14]
Label everything versus label uncertainty	Active selection reduces judge tokens and emphasizes informative traces.	Model uncertainty can omit confidently wrong examples.	[21], [33]
Classification versus ranking	Ranking matches selection and search.	Pairwise or Q-value targets may be noisy and less calibrated.	[9], [10], [15]
Discriminative versus generative verification	Generative verifiers can reason, critique, and scale at test time.	Higher token cost and possible unfaithful explanations.	[22], [25], [26], [32]
Parallel versus sequential test-time scaling	Parallel sampling improves coverage; sequential search exploits partial feedback.	More compute can amplify verifier errors or overthinking.	[34], [13], [7]
Static versus dynamic criteria	Dynamic criteria support cross-domain reward modeling.	Criterion generation and selection introduce another source of judge error.	[31], [19]
First-error versus reflective labels	Recovery-aware labels preserve valid reasoning after a mistake.	Context-sensitive labeling requires capable judges and longer traces.	[18], [17]
Neural versus deterministic evidence	Tools and rules reduce judge hallucination and reward hacking.	Many tasks lack executable steps or complete rule systems.	[29], [30]
Surface-text versus latent verification	Internal activations can expose correctness missed by textual self-judgment.	Requires white-box access and domain-matched experience.	[23]
External verification versus self-verification	Self-verification can be trained jointly with solving.	Self-judgment may become superficial without verifiable rewards.	[24], [28]
Inference rewards versus RL rewards	Dense rewards can improve sample efficiency.	Aggregation errors create reward hacking.	[11], [16]

Table 8. Recurring design tensions in verifier-guided multi-step reasoning. The same verifier can behave differently depending on whether it is used for reranking, search, critique, self-training, or reinforcement learning.

A useful synthesis is to distinguish methodological progress from field readiness. Methodologically, the literature now offers human labels, rollout estimates, confidence differences, tree-search values, pairwise preferences, implicit rewards, generative critiques, dynamic criteria, hidden-state distances, and deterministic constraint checks. In terms of readiness, however, several conditions are still missing. Verifiers are often evaluated on the same narrow domains that shaped their training data; uncertainty is rarely a calibrated output even when it is used for data selection; reflective recovery is absent from most first-error datasets; white-box and tool-grounded methods have restricted access assumptions; the same scalar reward is reused for reranking, search, and RL even though these consumers require different semantics; and compute budgets are usually fixed rather than allocated according to difficulty or confidence. The following discussion therefore treats each design tension not only as a comparison among papers, but also as evidence of an unresolved need.

One major lesson is that the meaning of a process reward is contextual. In PRM800K, the target is human step correctness. In Math-Shepherd and OmegaPRM, it is recoverability under sampled completions. AutoPSV labels relative changes in predicted outcome confidence. PQM orders Q-values, while Rewarding Progress estimates advantage-like improvement. Reflection-aware supervision labels whether a step propagates or terminates an earlier error. GroundedPRM mixes executable step fidelity with global success, and VPRM checks structured compliance with deterministic rules (Eq. 11). CLUE does not learn a process reward at all; it measures geometric similarity to successful experience. These quantities are related, but they are not interchangeable. A scalar trained for one use can fail when inserted into another, which explains why a verifier that improves best-of-N may still fail process diagnosis or RL optimization.

A second lesson is that verifier-guided test-time compute should be adaptive. Best-of-N is attractive because it is simple and parallel, but its performance depends on base-policy coverage and reward-model errors. MCTS and beam search exploit local verifier scores, but they can prune good branches if the verifier is

wrong. Hybrid TTS suggests that combining parallel and sequential methods at the step level can be more effective than using either alone, while InferenceTimePessimism shows that uncertainty-aware selection can avoid non-monotonic degradation. Dyve demonstrates adaptation inside the judge by reserving long analyses for selected steps, and CLUE demonstrates that some reranking can use a deterministic geometric statistic rather than another generated critique. These results point toward systems that estimate when to sample, search, refine, invoke a tool, think slowly, or stop.

A third lesson concerns verifier evidence and architecture. Discriminative PRMs are efficient and easy to plug into search, but they compress complex judgments into a scalar. Generative verifiers show that difficult checking benefits from explicit derivation, but they shift the compute bottleneck to the judge. Latent-state verification avoids textual judging but is limited to models whose activations are accessible. Tool- and rule-grounded verifiers offer stronger evidence but only for formalizable steps. The appropriate architecture is therefore conditional on what evidence exists, what access is available, and how frequently the verifier must run.

A fourth lesson is that supervision efficiency is part of verifier quality. AutoPSV reduces the number of generated continuation tokens by reusing an outcome verifier's confidence trajectory, while ActPRM reduces calls to an expensive annotator by selecting uncertain traces. GroundedPRM reports strong results from only 40K retained samples because its labels combine structured search and tool checks. These findings argue against comparing process datasets only by size. A more informative comparison reports annotation tokens, tool or judge calls, retained-sample rate, label agreement, and downstream performance under a matched data budget.

Benchmarking has become a central issue. Table 9 summarizes the main evaluation views. Answer accuracy is necessary but insufficient. Best-of-N accuracy measures whether a verifier helps select correct final answers, but may hide flawed reasoning. First-error localization evaluates causal process understanding, while PRMBench-style metrics test sensitivity to varied error types and false positives. RL evaluations test whether the reward can train a policy without being exploited. A mature evaluation suite should include all of these views because each exposes a different failure mode.

6.1 Cross-cutting research gaps

The reviewed literature suggests six gaps that cut across otherwise different verifier families. First, **semantic alignment** remains incomplete: local correctness, recoverability, Q-value, progress, and preference are distinct targets, yet their scores are often compared as if they were interchangeable. Second, **calibration under shift** is underdeveloped; a verifier can be confidently wrong when the generator, domain, prompt format, or reasoning style changes. Third, **recovery-aware evaluation** is missing from many process datasets, which makes a correct self-correction look like continued error. Fourth, **cost accounting** is fragmented: papers report training labels, judge calls, tokens, FLOPs, or latency in incompatible units. Fifth, **evidence grounding** is uneven: neural judges can reason flexibly but may hallucinate, while tools and deterministic rules are stronger but apply only to formalizable substeps. Sixth, **consumer-aware benchmarking** is still limited: a verifier used for reranking, search, diagnosis, and RL should not be judged by the same single metric.

These gaps motivate a system-level evaluation principle: every verifier paper should state (i) the property being verified, (ii) the intended consumer of the signal, (iii) the evidence available to the verifier, (iv) the uncertainty or abstention mechanism, (v) the compute and annotation budget, and (vi) at least one evaluation that directly measures the downstream use. This reporting template would make the literature substantially easier to reproduce and compare.

7 Open Problems and Future Directions

The review suggests that the field has entered a more mature stage. The central challenge is no longer simply to show that verifiers help, but to make verifier-guided reasoning reliable under distribution shift, large search budgets, policy optimization, and broader task domains. Table 10 summarizes the main shortcomings and possible solution directions. The table is not intended as a checklist of independent problems; rather, the rows

Evaluation view	What it measures	What it can miss	Examples
Final-answer accuracy	Whether the selected solution reaches the correct answer.	Invalid or unfaithful reasoning paths.	GSM8K, MATH, MATH500, AIME.
Best-of-N / reranking	Whether verifier scores improve selection among candidates.	Step-level false positives and reward overoptimization.	[3], [8], [15].
First-error localization	Whether the verifier finds the earliest erroneous step.	Broader qualities such as redundancy, simplicity, or sensitivity.	[18].
Fine-grained process diagnosis	Whether the verifier detects multiple error types and process-quality defects.	Generalization beyond textual reasoning.	[19].
Search scaling curves	Whether additional inference compute helps monotonically.	Training-time reward hacking.	[34], [13], [7].
Reflective recovery	Whether a verifier recognizes valid self-correction after an error.	First-error-only performance and non-reflective traces.	[17].
Structured coherence	Whether intermediate labels deterministically imply and agree with the final decision.	Open-ended reasoning without an explicit decision rule.	[30].
Label efficiency	Performance relative to annotation tokens, judge calls, or selected samples.	Raw accuracy at unconstrained labeling cost.	[21], [33], [29].
RL training stability	Whether process rewards improve the policy without exploitation.	Pure inference-time usefulness.	[11], [16], [28], [30].

Table 9. Evaluation protocols for verifier-guided reasoning. No single protocol is sufficient because verifiers are used for different downstream purposes.

interact. For instance, adaptive compute allocation requires calibrated uncertainty, and safe self-improvement requires evaluation protocols that can detect reward hacking before it is amplified by training.

The first open problem is reliable, calibrated supervision under distribution shift. Many PRMs are trained on mathematical traces from a limited set of generators, but are then used on harder problems, stronger or weaker policies, or out-of-domain tasks. ProcessBench and PRMBench show that this gap is substantial [18, 19]. ActPRM shows how model disagreement can reduce labeling cost, but an active learner can also ignore examples on which every head is confidently wrong [33]. CLUE similarly finds that fixed centroids degrade under domain and prompt mismatch [23]. A practical verifier should therefore report confidence, calibration error, and abstention behavior under generator, domain, and prompt shifts. Low confidence should preserve search diversity, trigger an independent tool or judge, reduce RL reward magnitude, or defer annotation rather than silently commit to a label.

The second open problem is adaptive allocation of test-time compute. Current systems often fix a budget for sampling, search, refinement, or verifier-side reasoning. Hybrid TTS shows that parallel and sequential scaling can be complementary, while best-of-N theory and scaling-flaw analyses show that more compute can eventually hurt [7, 15, 13]. Dyve provides an empirical fast–slow policy, but its routing is implicit in supervised generation and is not optimized for a global latency or accuracy budget [32]. A stronger controller would estimate problem difficulty, branch diversity, verifier uncertainty, external-tool availability, and marginal gain, then decide whether to use a scalar or latent check, request a long critique, sample another solution, refine a step, expand an MCTS branch, execute a tool, or stop.

The third open problem is aligning verifier semantics with downstream use and long-CoT dynamics. A verifier used for first-error localization should be trained and evaluated differently from one used for MCTS, best-of-N, or RL. Stop Summation shows that even a useful process reward can become harmful when aggregated

Unanswered need	Why current work is insufficient	General solution direction
Reliable and efficient labels	Human labels are costly, while rollout, confidence, and judge labels can reflect policy or judge bias rather than true step quality.	Combine small human audits, outcome checks, tools, consensus filtering, and active labeling; report cost and agreement with performance.
Calibrated verification	Search, active selection, and RL treat scores or disagreement as trustworthy, but overconfident errors are amplified.	Train calibrated uncertainty and abstention, stress-test prompt and domain shift, and use pessimistic selection when confidence is low.
Objective–consumer alignment	Correctness, recoverability, Q-value, advantage, preference, critique, constraints, and min-form reliability serve different algorithms.	Specify verifier targets jointly with first-error diagnosis, reranking, MCTS, refinement, or policy optimization.
Reflection-aware semantics	First-error truncation cannot represent a valid correction after an earlier mistake.	Annotate transition types and recovery spans, and evaluate error propagation, cessation, and return to a valid path.
Adaptive compute control	Fixed budgets for sampling, refinement, search, tools, or verifier reasoning ignore difficulty and marginal returns.	Route among cheap scores, latent checks, fast–slow critique, external tools, tree search, and early stopping.
Grounded domain extension	Deterministic rewards are reliable but require executable operations or complete decision rules.	Develop typed reasoning interfaces, modular verifiers, retrieval-backed evidence checks, and hybrid neural–symbolic fallback.
Robust process evaluation	Final accuracy and best-of-N hide flawed reasoning, while current process benchmarks remain math-heavy and first-error-centric.	Jointly test answers, first errors, recovery, coherence, calibration, label efficiency, scaling curves, and RL exploitability.
Safe self-improvement	Self-training and RL can reinforce blind spots, reduce diversity, or optimize reward artifacts.	Use independent verifiers, adversarial traces, held-out audits, uncertainty penalties, and periodic human or tool-grounded checks.

Table 10. A research agenda for verifier-guided multi-step reasoning. The main gap is not the absence of verifiers, but the lack of system-level guarantees that connect verifier training, uncertainty, compute allocation, and downstream use.

incorrectly in reinforcement learning [16]. Beyond the First Error shows that binary prefix-monotone labels can also misrepresent a model that notices and repairs its own mistake [17]. The field needs clearer theory for choosing among local correctness, recoverability, Q-value, advantage, preference, constraints, dynamic criteria, and transition-aware reflection. Search papers should report ranking and pruning behavior; RL papers should report exploitability; diagnostic papers should report false positives and recovery detection; and reflective models should report whether the verifier recognizes a genuine return to a valid reasoning state.

The fourth open problem is safe self-improvement. ReST-MCTS*, rStar-Math, critique-model training, and RISE show that verifier-guided systems can generate better training data for themselves [6, 12, 24, 28]. Yet self-training can amplify blind spots. Active selection can repeatedly favor one uncertainty pattern, search can remove diversity, and a policy can learn to exploit the reward used to select its own data. Future systems should include independent evaluation channels, adversarial reasoning traces, held-out process audits, and exploration guarantees. Grounded or deterministic checks are valuable when available, but should not be conflated with complete safety because tools and guidelines can themselves be incomplete.

The fifth open problem is moving beyond math-heavy textual reasoning without losing verifiability. DG-PRM is a step toward dynamic criteria, and VPRM demonstrates deterministic process rewards in medical evidence assessment [31, 30]. The contrast is revealing: domain-specific rules can make reasoning transparent and rewardable, but only after the process has been represented as explicit, verifier-compatible steps. Broader domains require typed reasoning interfaces that expose claims, evidence, operations, and decision transitions;

modular verifiers that combine symbolic tools, retrieval, simulation, and human preference; and a fallback policy for steps that cannot be checked. This is especially important in science, medicine, law, and agents, where a plausible conclusion can rest on omitted evidence or invalid constraints.

The sixth open problem is lifecycle efficiency and controllability. Cost arises before deployment through labeling and during deployment through repeated judging. AutoPSV and ActPRM reduce the first cost using confidence trajectories and active selection, while Dyve and CLUE reduce or restructure the second using fast–slow generation and non-parametric latent scoring [21, 33, 32, 23]. These savings are not directly comparable because they count different resources. Future work should report a common cost ledger: generated annotation tokens, judge and tool calls, accelerator time, retained examples, verifier tokens, memory overhead for hidden states, and end-to-end accuracy or F1. Such reporting would support a genuine compute-optimal verifier pipeline rather than isolated efficiency claims.

The seventh open problem is benchmark validity. Answer accuracy, best-of-N, first-error localization, PRMBench-style diagnosis, scaling curves, RL stability, reflective recovery, structured coherence, and label efficiency reveal different properties. None is sufficient alone. A mature suite should test whether a verifier helps its intended consumer, detects the first causal error, recognizes a valid correction after that error, remains calibrated under stronger or weaker generators, agrees with executable or rule-based evidence where available, and can be optimized without reward hacking. Evaluation should therefore move from single-number leaderboards toward multi-view stress tests with matched compute and annotation budgets.

8 Conclusion

Verifier-guided multi-step reasoning has developed from a simple selection mechanism into a broad research program. Early work showed that process supervision can outperform outcome supervision for selecting mathematical solutions. Later work made process supervision scalable through automated rollouts, search, preference learning, Q-value rankings, and implicit rewards. The newest work adds confidence-derived and actively selected labels, reflective post-error supervision, hidden-state verification, execution-grounded and rule-verifiable rewards, dynamic fast–slow judging, and structured non-mathematical applications. Together, the reviewed papers show that verification is not a single module but a set of evidence, representation, training, inference, and evaluation choices.

The strongest conclusion from the literature, reinforced by the matched empirical comparison in Section 5, is that there is no universal winner and that verifier quality must be judged relative to evidence and use. Discriminative PRMs offer the best speed–accuracy balance for latency-sensitive deployment; generative PRMs achieve the highest verification accuracy at substantially higher inference cost; and verifier-guided search is most useful at small or moderate compute budgets, not necessarily at very large N. A verifier may be good for best-of-N but weak at first-error localization; accurate on first-error traces but wrong after genuine self-correction; useful for reranking but unsafe as an RL reward; robust with domain-matched hidden-state experience but brittle under prompt shift; objective for tool-checkable steps but silent about conceptual errors; accurate when allowed to think but too expensive for dense search. Apparent disagreements often reflect different reward semantics, access assumptions, and consumption algorithms. A reliable system should therefore co-design the generator, evidence source, verifier, search procedure, aggregation rule, uncertainty mechanism, and evaluation protocol.

Recent additions sharpen this message. AutoPSV and ActPRM show that process data should be judged by information and annotation cost, not only volume. CLUE shows that correctness can be readable from a frozen reasoner's internal trajectory, while also exposing the price of white-box and domain-matched access. GroundedPRM and VPRM show two forms of verifiable process evidence: executable intermediate operations and deterministic structured decision rules. Dyve demonstrates learned fast–slow verification, and Beyond the First Error shows that reflective long reasoning invalidates the standard assumption that correctness ends permanently at the first mistake. Combined with hybrid test-time scaling, dynamic criteria, reasoning PRMs, process benchmarks, and best-of-N theory, these findings point toward verifier-guided reasoners that are adaptive, grounded, recovery-aware, calibrated, and cost-aware.

The broader lesson is that current verifiers improve accuracy without yet providing a complete route to deployable reasoning systems. Within the scope of the reviewed corpus, the evidence is strongest for mathematical and other structured tasks with an explicit correctness signal; claims about general-purpose reasoning should therefore be treated as a research hypothesis rather than an established result. Reliable labels, calibrated uncertainty, post-error recovery, domain-general criteria, common cost accounting, and safety under self-improvement remain unresolved. The general path forward is to treat verification as a system property: use the strongest available evidence for each step, combine cheap and expensive judges, preserve uncertainty and diversity, adapt compute to difficulty, evaluate both error and recovery, and keep independent or deterministic checks in the loop whenever a model learns from its own verified outputs.

References

[1] Q. Zhang, F. Lyu, Z. Sun, L. Wang, W. Zhang, W. Hua, H. Wu, Z. Guo, Y. Wang, N. Muennighoff, I. King, X. Liu, and C. Ma, "A survey on test-time scaling in large language models: What, how, where, and how well?" *arXiv preprint arXiv:2503.24235*, 2025. [Online]. Available: https://arxiv.org/abs/2503.24235

[2] Z.-Z. Li, D. Zhang, M.-L. Zhang, J. Zhang, Z. Liu, Y. Yao, H. Xu, J. Zheng, P.-J. Wang, X. Chen *et al.*, "From system 1 to system 2: A survey of reasoning large language models," *arXiv preprint arXiv:2502.17419*, 2025. [Online]. Available: https://arxiv.org/abs/2502.17419

[3] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, "Let's verify step by step," *arXiv preprint arXiv:2305.20050*, 2023. [Online]. Available: https://arxiv.org/abs/2305.20050

[4] P. Wang, L. Li, Z. Shao, R. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui, "Math-shepherd: Verify and reinforce llms step-by-step without human annotations," *arXiv preprint arXiv:2312.08935*, 2023. [Online]. Available: https://arxiv.org/abs/2312.08935

[5] L. Luo, Y. Liu, R. Liu, S. Phatale, M. Guo, H. Lara, Y. Li, L. Shu, Y. Zhu, L. Meng, J. Sun, and A. Rastogi, "Improve mathematical reasoning in language models by automated process supervision," *arXiv preprint arXiv:2406.06592*, 2024. [Online]. Available: https://arxiv.org/abs/2406.06592

[6] D. Zhang, S. Zhoubian, Z. Hu, Y. Yue, Y. Dong, and J. Tang, "Rest-mcts*: Llm self-training via process reward guided tree search," *arXiv preprint arXiv:2406.03816*, 2024. [Online]. Available: https://arxiv.org/abs/2406.03816

[7] K. Chang, Y. Shi, C. Wang, H. Zhou, C. Hu, X. Liu, Y. Luo, Y. Ge, T. Xiao, and J. Zhu, "Step-level verifier-guided hybrid test-time scaling for large language models," *arXiv preprint arXiv:2507.15512*, 2025. [Online]. Available: https://arxiv.org/abs/2507.15512

[8] Z. Zhang, C. Zheng, Y. Wu, B. Zhang, R. Lin, B. Yu, D. Liu, J. Zhou, and J. Lin, "The lessons of developing process reward models in mathematical reasoning," *arXiv preprint arXiv:2501.07301*, 2025. [Online]. Available: https://arxiv.org/abs/2501.07301

[9] M. He, Y. Shen, W. Zhang, Z. Tan, and W. Lu, "Advancing process verification for large language models via tree-based preference learning," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024, pp. 2086–2099. [Online]. Available: https://arxiv.org/abs/2407.00390

[10] W. Li and Y. Li, "Process reward model with q-value rankings," in *International Conference on Learning Representations*, 2025. [Online]. Available: https://arxiv.org/abs/2410.11287

[11] A. R. Setlur, C. Nagpal, A. Fisch, X. Geng, J. Eisenstein, R. Agarwal, A. Agarwal, J. Berant, and A. Kumar, "Rewarding progress: Scaling automated process verifiers for llm reasoning," *arXiv preprint arXiv:2410.08146*, 2024. [Online]. Available: https://arxiv.org/abs/2410.08146

[12] X. Guan, L. L. Zhang, Y. Liu, N. Shang, Y. Sun, Y. Zhu, F. Yang, and M. Yang, "rstar-math: Small llms can master math reasoning with self-evolved deep thinking," *arXiv preprint arXiv:2501.04519*, 2025. [Online]. Available: https://arxiv.org/abs/2501.04519

[13] F. Yu, Y. Li, and B. Wang, "Scaling flaws of verifier-guided search in mathematical reasoning," *arXiv preprint arXiv:2502.00271*, 2025. [Online]. Available: https://arxiv.org/abs/2502.00271

[14] L. Yuan, W. Li, H. Chen, G. Cui, N. Ding, K. Zhang, B. Zhou, Z. Liu, and H. Peng, "Free process rewards without process labels," *arXiv preprint arXiv:2412.01981*, 2024. [Online]. Available: https://arxiv.org/abs/2412.01981

[15] A. Huang, A. Block, Q. Liu, N. Jiang, A. Krishnamurthy, and D. J. Foster, "Is best-of-N the best of them? coverage, scaling, and optimality in inference-time alignment," *arXiv preprint arXiv:2503.21878*, 2025. [Online]. Available: https://arxiv.org/abs/2503.21878

[16] J. Cheng, G. Xiong, R. Qiao, L. Li, C. Guo, J. Wang, Y. Lv, and F.-Y. Wang, "Stop summation: Min-form credit assignment is all process reward model needs for reasoning," *arXiv preprint arXiv:2504.15275*, 2025. [Online]. Available: https://arxiv.org/abs/2504.15275

[17] Z. Yang, C. He, X. Shi, L. Li, Q. Yin, S. Deng, and D. Jiang, "Beyond the first error: Process reward models for reflective mathematical reasoning," *arXiv preprint arXiv:2505.14391*, 2025. [Online]. Available: https://arxiv.org/abs/2505.14391

[18] C. Zheng, Z. Zhang, B. Zhang, R. Lin, K. Lu, B. Yu, D. Liu, J. Zhou, and J. Lin, "Processbench: Identifying process errors in mathematical reasoning," *arXiv preprint arXiv:2412.06559*, 2024. [Online]. Available: https://arxiv.org/abs/2412.06559

[19] M. Song, Z. Su, X. Qu, J. Zhou, and Y. Cheng, "PRMBench: A fine-grained and challenging benchmark for process-level reward models," *arXiv preprint arXiv:2501.03124*, 2025. [Online]. Available: https://arxiv.org/abs/2501.03124

[20] Z. Jia, A. Rakhlin, and T. Xie, "Do we need to verify step by step? rethinking process supervision from a theoretical perspective," *arXiv preprint arXiv:2502.10581*, 2025. [Online]. Available: https://arxiv.org/abs/2502.10581

[21] J. Lu, Z. Dou, H. Wang, Z. Cao, J. Dai, Y. Wan, and Z. Guo, "AutoPSV: Automated process-supervised verifier," in *Advances in Neural Information Processing Systems*, 2024. [Online]. Available: https://arxiv.org/abs/2405.16802

[22] L. Zhang, A. Hosseini, H. Bansal, M. Kazemi, A. Kumar, and R. Agarwal, "Generative verifiers: Reward modeling as next-token prediction," in *International Conference on Learning Representations*, 2025. [Online]. Available: https://arxiv.org/abs/2408.15240

[23] Z. Liang, R. Li, Y. Zhou, L. Song, D. Yu, X. Du, H. Mi, and D. Yu, "Your reasoning model is secretly a reward model: Optimization-free verification from experience," in *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026, pp. 17 358–17 372. [Online]. Available: https://aclanthology.org/2026.acl-long.788/

[24] Z. Xi, D. Yang, J. Huang, J. Tang, G. Li, Y. Ding, W. He, B. Hong, S. Do, W. Zhan, X. Wang, R. Zheng, T. Ji, X. Shi, Y. Zhai, R. Weng, J. Wang, X.-Y. Cai, T. Gui, Z. Wu, Q. Zhang, X. Qiu, X. Huang, and Y.-G. Jiang, "Enhancing llm reasoning via critique models with test-time and training-time supervision," *arXiv preprint arXiv:2411.16579*, 2024. [Online]. Available: https://arxiv.org/abs/2411.16579

[25] J. Zhao, R. Liu, K. Zhang, Z. Zhou, J. Gao, D. Li, J. Lyu, Z. Qian, B. Qi, X. Li, and B. Zhou, "Genprm: Scaling test-time compute of process reward models via generative reasoning," *arXiv preprint arXiv:2504.00891*, 2025. [Online]. Available: https://arxiv.org/abs/2504.00891

[26] M. Khalifa, R. Agarwal, L. Logeswaran, J. Kim, H. Peng, M. Lee, H. Lee, and L. Wang, "Process reward models that think," *arXiv preprint arXiv:2504.16828*, 2025. [Online]. Available: https://arxiv.org/abs/2504.16828

[27] S. She, J. Liu, Y. Liu, J. Chen, X. Huang, and S. Huang, "R-PRM: Reasoning-driven process reward modeling," *arXiv preprint arXiv:2503.21295*, 2025. [Online]. Available: https://arxiv.org/abs/2503.21295

[28] X. Liu, T. Liang, Z. He, J. Xu, W. Wang, P. He, Z. Tu, H. Mi, and D. Yu, "Trust, but verify: A self-verification approach to reinforcement learning with verifiable rewards," *arXiv preprint arXiv:2505.13445*, 2025. [Online]. Available: https://arxiv.org/abs/2505.13445

[29] Y. Zhang, Y. Wu, H. Zhang, W. Li, H. Chen, J. Wu, G. Li, Z. Han, and V. Tresp, "GroundedPRM: Tree-guided and fidelity-aware process reward modeling for step-level reasoning," *arXiv preprint arXiv:2510.14942*, 2025. [Online]. Available: https://arxiv.org/abs/2510.14942

[30] M. Pronesti, A. Belz, and Y. Hou, "Beyond outcome verification: Verifiable process reward models for structured reasoning," *arXiv preprint arXiv:2601.17223*, 2026. [Online]. Available: https://arxiv.org/abs/2601.17223

[31] Z. Yin, Q. Sun, Z. Zeng, Q. Cheng, X. Qiu, and X. Huang, "Dynamic and generalizable process reward modeling," *arXiv preprint arXiv:2507.17849*, 2025. [Online]. Available: https://arxiv.org/abs/2507.17849

[32] J. Zhong, Z. Li, Z. Xu, X. Wen, and Q. Xu, "Dyve: Thinking fast and slow for dynamic process verification," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025, pp. 22 320–22 333. [Online]. Available: https://aclanthology.org/2025.emnlp-main.1136/

[33] K. Duan, Z. Liu, X. Mao, T. Pang, C. Chen, Q. Chen, M. Q. Shieh, and L. Dou, "Efficient process reward model training via active learning," *arXiv preprint arXiv:2504.10559*, 2025. [Online]. Available: https://arxiv.org/abs/2504.10559

[34] C. V. Snell, J. Lee, K. Xu, and A. Kumar, "Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning," in *International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=4FWAwZtd2n

A Additional Details

This appendix is left available for extended benchmark tables and ablation details.