A Survey on Recursive Reasoning Models

Aref Mousavi* *aref.mousavi.ac@gmail.com*
Nima Zare* *nimazare.dev@gmail.com*
Amir Hossein Movahedi* *movahedisefat03@gmail.com*
Mahshid Dehghani† *mahshid.dehghani5@gmail.com*
Muhammad Shirkhani† *muhammad.shirkhani@gmail.com*
Danial Parnian *danielparnian@gmail.com*

Abstract

Recursive reasoning models offer a compact alternative to chain-of-thought-style scaling for structured reasoning. Rather than externalizing intermediate computation into long token sequences, they repeatedly apply parameter-shared learned transitions to persistent latent states, candidate answers, or reasoning trajectories. This survey develops an operational definition of recursive reasoning models and organizes recent work into five taxonomy branches: four mechanism-centered families (architectural foundations, dynamical-systems approaches, trajectory-search methods, and training optimization) plus an *Application Extensions* branch for work whose primary contribution is transferring recursive latent-computation mechanisms to a new domain or system. Across Sudoku, maze solving, ARC-AGI, and related algorithmic benchmarks, current evidence suggests that recursive computation can enable parameter-efficient reasoning and support test-time scaling through greater recursion depth or trajectory width. Cross-paper synthesis, however, shows that these gains are conditional: current evidence does not establish hierarchical recurrence as necessary, more recursive depth is not uniformly beneficial, and width scaling helps only when exploration and candidate selection remain well aligned. Meaningful comparison therefore remains difficult without standardized reporting of training setup, recursion budget, number of trajectories, selection or verifier use, pass@k, and exact-solve accuracy. We identify stable depth extrapolation, long-horizon credit assignment, trajectory selection, and benchmark comparability as central challenges for building reliable recursive reasoners.

1 Introduction

Large language models have demonstrated strong capabilities in natural language, code generation, and tool use, yet their reliability on structured reasoning problems remains uneven. Tasks such as Sudoku, maze solving, ARC-AGI, and synthetic algorithmic benchmarks often require repeated constraint propagation, hypothesis revision, local correction, and global consistency checking (Chollet, 2019). Within a conventional forward pass, a standard Transformer applies a fixed stack of layers rather than adapting its computational depth to the difficulty of an individual instance (Vaswani et al., 2017; Merrill & Sabharwal, 2023). Chain-of-Thought (CoT) prompting provides one way to allocate additional computation by generating intermediate reasoning tokens. However, this strategy requires additional autoregressive decoding, and the resulting textual trace need not faithfully reflect the computation responsible for the final answer (Wei et al., 2022; Lanham et al., 2023).

*These authors contributed equally and share first authorship.
†These authors contributed equally.

Recursive reasoning models offer a complementary approach. Rather than increasing computation primarily through a longer output sequence, they repeatedly apply parameter-shared learned transitions to a persistent internal state. This state may represent a latent reasoning variable, an intermediate answer, a collection of candidate solutions, or a trajectory within a learned dynamical system. Reusing the same transition decouples parameter count from executed computational depth, allowing additional internal computation without introducing a distinct parameter block at every recursive step. This principle connects recent recursive reasoners to earlier work on adaptive computation, recurrent-depth Transformers, and implicit neural layers (Graves, 2016; Dehghani et al., 2019; Bai et al., 2019). More recently, the Hierarchical Reasoning Model (HRM) introduced interacting high- and low-level recurrent modules, whereas the Tiny Recursive Model (TRM) showed that competitive structured-reasoning performance can also emerge from a substantially simpler shared-network recursion (Wang et al., 2025; Jolicoeur-Martineau, 2025).

Throughout this survey, we use the term *recursive* in a controlled architectural sense. A recursive reasoning model contains an internal learned loop that repeatedly updates a persistent problem-solving state within a single inference episode. Recursion in this sense is distinct from ordinary autoregressive token generation, sequence-time recurrence, repeated prompting, external agent calls, and tree-search wrappers that invoke a model as a black box. These mechanisms may all involve iteration, but iteration alone does not place a system in the RRM class. They also raise different questions about parameter sharing, credit assignment, stability, halting, verification, and inference-time scaling. Under the terminology adopted in this survey, recursive reasoning models occupy the part of the broader loop-model landscape in which repeated learned computation serves as the mechanism for constructing, revising, scoring, or selecting a solution to a fixed reasoning instance. Systems that reuse or adapt the same recursive latent-computation primitive inside broader language, generative, or neuro-symbolic pipelines are treated separately as application extensions. Inclusion in this branch does not by itself imply that the complete system satisfies the strict fixed-instance RRM definition adopted below.

The rapid expansion of this literature has produced a diverse collection of architectures, training objectives, and inference procedures without a fully shared vocabulary for comparing them. One direction studies architectural foundations, including hierarchical and single-module backbones and more expressive repeated operators (Wang et al., 2025; Jolicoeur-Martineau, 2025; Gao et al., 2025). Another interprets recursive inference as a dynamical system and investigates fixed points, attractors, stability, and convergence (Es'kin & Smorkalov, 2026; Huang et al., 2026; Movahedi et al., 2026; Ren & Liu, 2026). A third direction introduces trajectory search by perturbing, sampling, guiding, or selecting among multiple reasoning paths (Baek et al., 2026b; Cameron et al., 2026; Sghaier et al., 2026; Corbett et al., 2026). A fourth focuses on training optimization through supervision schemes, depth curricula, efficient unrolling, and adaptation (Qasim & Zhang, 2025; Asadulaev et al., 2025; Hakimi, 2026; Çakar et al., 2026). A fifth branch isolates application-oriented extensions in which the main novelty is transferring or integrating a recursive latent-computation mechanism into language pretraining, exact symbolic solving, or image generation (Wang et al., 2026; Bertram et al., 2026; Esmaeilzadeh et al., 2026). These directions interact, but the first four modify different components of recursive computation, whereas the Application Extensions branch records where the same general primitive is being deployed.

A further difficulty is that reported results are often not directly comparable. Existing studies differ in parameter count, training data, augmentation, recursion depth, number of sampled trajectories, selection or verifier use, and evaluation protocol. Increasing recursive depth and increasing trajectory width are distinct forms of test-time computation, while pass@1, pass@k, majority-vote accuracy, and exact-solve accuracy measure related but different capabilities. Consequently, a method may improve candidate coverage without improving top-ranked selection, or obtain a stronger reported score primarily through a larger inference budget rather than a better recursive transition (Roye-Azar et al., 2025; Wang & Reid, 2026; Sghaier et al., 2026; Corbett et al., 2026). A useful survey must therefore compare not only final accuracy, but also the computational and selection mechanisms through which that accuracy is obtained.

This survey makes four contributions. First, synthesizing the lineage from adaptive computation and recurrent-depth Transformers through recurrent algorithm-learning systems and implicit fixed-point models, and using GRAM's probabilistic trajectory formulation as a modern stochastic instantiation (Graves, 2016; Dehghani et al., 2019; Schwarzschild et al., 2021; Bansal et al., 2022; Bai et al., 2019; Baek et al., 2026b),

we propose an operational definition of recursive reasoning models based on a persistent internal state, parameter-shared transitions, and answer-coupled computation. This definition separates recursive latent deliberation from ordinary sequence recurrence, external search, and repeated full-model invocation. Second, we position recursive reasoning within the broader loop-model landscape while restricting the survey's primary scope to learned neural loops for fixed-instance structured reasoning. Third, we organize recent work into the taxonomy shown in Figure 3: four mechanism-centered families (*architectural foundations*, *dynamical systems*, *trajectory search*, and *training optimization*) together with a separate *Application Extensions* branch for domain and system extensions. Fourth, we synthesize the central design trade-offs and evaluation challenges in the field, including recursive depth versus trajectory width, deterministic convergence versus stochastic exploration, generic operators versus task-specific inductive biases, and candidate generation versus reliable selection. We further make the evaluation contract explicit by separating benchmark semantics, candidate coverage, selection quality, attempt budget, and end-to-end compute, and distill the reviewed evidence into survey-level findings that distinguish supported, contradicted, and still-unestablished claims.

Relation to prior surveys. Recent surveys have mapped the broader latent-reasoning landscape from complementary perspectives. (Chen et al., 2025) organize latent Chain-of-Thought methods into token-wise and layer-wise computation, covering both hidden-state reasoning and recurrent-depth approaches. (Zhu et al., 2025) take an even broader view that includes activation-based recurrence, hidden-state propagation, interpretability, and diffusion- or equilibrium-inspired routes toward effectively unbounded latent computation. Test-time scaling has also been surveyed at the level of general LLM inference, including sampling, search, verification, and adaptive compute allocation (Zhang et al., 2025). Our scope is deliberately narrower. We center the internal, parameter-shared loop itself and ask how a persistent answer-coupled state is initialized, updated, stabilized, broadened into multiple trajectories, supervised, halted, and evaluated. This narrower boundary lets us separate recursive depth from trajectory width, distinguish candidate coverage from candidate selection, and compare fixed-point, stochastic, and structured-transition views under one operational definition. We therefore position this survey as a focused account of recursive reasoning models rather than as another general survey of latent reasoning or test-time scaling.

Scope and review protocol. This article is a structured narrative survey with a literature snapshot through **July 31, 2026**. Coverage was assembled by starting from the foundational recurrent-computation lineage and the modern HRM/TRM family, then expanding with targeted searches around terms such as *recursive reasoning*, *recurrent depth*, *looped Transformer*, *latent reasoning*, *fixed point*, *attractor*, *trajectory search*, and *test-time scaling*, together with paper-level reference and related-work tracing. A work is included in the core review when its learned inference mechanism satisfies the fixed-instance, persistent-state, answer-coupling, and parameter-sharing conditions formalized in Section 2.3. Closely related systems whose main contribution is transferring the same primitive to a different interface are kept in the Application Extensions branch. Purely external prompting or search loops, ordinary sequence-time recurrence, token-only Chain-of-Thought, and untied depth scaling are excluded from the core class unless they are needed to clarify a boundary or historical connection. Because this is a structured narrative review rather than a systematic review, and the area is unusually fast-moving, the cutoff should be read as a reproducible snapshot rather than a guarantee that every adjacent loop-model paper is captured.

Our goal is not to construct a leaderboard from heterogeneous results, but to identify the computational contracts that distinguish existing approaches, clarify what their empirical evidence supports, and expose the conditions under which additional recursive computation is useful. We therefore discuss CoT, symbolic search, diffusion, generic recurrent sequence models, and other looped architectures primarily when they clarify the boundary of the field. Across the reviewed literature, we identify stable depth extrapolation, long-horizon credit assignment, trajectory selection, mechanistic understanding, and compute-matched evaluation as central challenges for building reliable recursive reasoners.

To ensure mathematical consistency across the reviewed literature, a unified summary of all symbols, variables, and mathematical definitions used throughout this survey is provided in Appendix A (Table 9).

The remainder of the survey is organized as follows. Section 2 traces the computational lineage of recursive reasoning, positions it within the broader loop-model landscape, and introduces the formal definition used

throughout the paper. Section 3 presents the taxonomy, and Section 4 reviews representative methods within each family. Section 5 summarizes the main benchmarks, clarifies the meaning of common metrics, and organizes the protocol-sensitive empirical evidence. Section 6 then compares the mechanisms, analyzes their design trade-offs, and distills the strongest cross-paper findings from the reviewed evidence. Section 7 discusses open problems and directions for future work, and Section 8 states the scope and evidential limitations of the survey before the conclusion.

2 Background

Structured reasoning as iterative refinement. Many benchmarks used in the recursive-reasoning literature evaluate globally constrained structured outputs rather than local predictive accuracy alone. In Sudoku, a solution must simultaneously satisfy row, column, and box constraints. In maze solving, the predicted path must be valid under the spatial connectivity of the maze. In ARC-AGI (Chollet, 2019), a model must infer a transformation rule from a small set of demonstrations and apply it consistently to a held-out grid. In each setting, an intermediate prediction may satisfy local regularities while remaining globally incorrect. These tasks therefore provide natural testbeds for models that iteratively revise an internal state or candidate solution before producing a final output.

2.1 From Recurrent Computation to Recursive Reasoning

The central mechanisms used by modern recursive reasoning models did not appear all at once. They emerge from several earlier lines of work that separated three ideas now combined in RRMs: allocating variable amounts of computation, reusing parameters across computational depth, and interpreting repeated updates through convergence or equilibrium. Tracing this lineage is important because none of these ingredients alone is sufficient to define recursive *reasoning*; the distinction lies in how the recurrent computation is coupled to solving a fixed problem instance.

Adaptive computation and learned halting. Adaptive Computation Time (ACT) made computational budget itself a learned quantity by allowing a recurrent network to decide how many internal updates to execute before emitting an output (Graves, 2016). This idea is directly relevant to modern RRM controllers and halting heads: difficult instances need not receive the same number of updates as easy ones. ACT, however, is a general mechanism for variable computation in recurrent networks. It does not by itself specify that the repeated state constitutes a problem-solving trajectory for one fixed structured reasoning instance.

Recurrent depth and iterative representation refinement. Universal Transformers moved parameter sharing into the depth axis of the Transformer by repeatedly applying a self-attentive transition to the evolving representation (Dehghani et al., 2019). This established an important architectural precursor to current RRMs: effective depth can increase while the learned transition parameters remain tied. The resulting recurrence also supports iterative refinement of a persistent representation. Yet recurrent depth is still broader than recursive reasoning; a weight-tied Transformer is not an RRM under our definition unless that recurrent axis serves as the answer-coupled internal deliberation process for the same problem instance.

Recurrent algorithm learning and depth extrapolation. A closer bridge to today's structured RRMs appeared in recurrent networks trained to learn iterative algorithms on problems such as mazes and other algorithmic tasks. (Schwarzschild et al., 2021) showed that a shared recurrent module trained on easier instances can sometimes solve harder instances simply by executing additional recurrences at test time. (Bansal et al., 2022) then identified *overthinking*: a recurrent solver can deteriorate when run substantially longer than during training, and proposed input recall together with progressive training to make the learned update more repeatable across depth. More recent Deep Thinking work shows that stability of such iterative algorithms is itself a design problem, with Lipschitz-constrained recurrence used to control long-horizon dynamics (Bear et al., 2024). At language-model scale, (Geiping et al., 2025) showed that a 3.5B-parameter model could use a repeatedly unrolled shared block as a test-time compute axis, demonstrating that recurrent depth can remain useful well beyond small algorithm-learning systems. (Saunshi et al., 2025) complemented this empirical direction with controlled and theoretical evidence that looped Transformers can use effective

depth for iterative reasoning and, under their construction, simulate multi-step Chain-of-Thought computation in latent form. We treat these works as close precursors rather than core structured RRMs because their primary setting is language modeling or general looped-Transformer reasoning, but together they provide an important bridge from parameter-tied depth to modern latent test-time scaling. This line is especially close to modern RRMs because it treats extra recurrent depth as a test-time problem-solving resource and directly exposes the depth-extrapolation and stability questions that recur throughout the present literature.

Equilibrium and fixed-point views. Deep Equilibrium Models (DEQs) provided a complementary interpretation of weight-tied depth by treating the representation as the fixed point of a learned transformation and solving for that equilibrium directly (Bai et al., 2019). This perspective supplied a mathematical language for effective infinite depth, convergence, and implicit differentiation that later reappears in RRM work on attractors, residual-based stopping, and stability. At the same time, an equilibrium is only a dynamical property: convergence of the latent state does not establish that the decoded answer is correct. Modern recursive reasoners therefore inherit the fixed-point lens without identifying reasoning quality with convergence alone.

The modern shift: recurrence as answer-coupled deliberation. The HRM/TRM wave can be understood as bringing these recurrent-computation ideas into a more specialized role: the recurrent axis is used to construct and revise a solution to a fixed reasoning problem rather than merely to process sequence positions or define an implicit layer (Wang et al., 2025; Jolicoeur-Martineau, 2025). The persistent state can carry latent features, intermediate answer estimates, confidence, or multiple candidate hypotheses, and additional recursive computation can change the final prediction. Recent stochastic formulations such as GRAM extend this view from a single recurrent path to a distribution over latent reasoning trajectories (Baek et al., 2026b). The novelty of the contemporary RRM literature is therefore not recurrence itself, but the use of parameter-shared recurrent depth as an explicit, answer-coupled inference resource whose depth, width, stopping behavior, stability, and selection can be studied independently. This historical synthesis motivates the stricter operational boundary introduced below.

2.2 Position in the Broader Loop-Model Landscape

The lineage above places recursive reasoning models inside a broader family of looped neural architectures in which a learned layer, module, or transition is reused within a single inference episode. Adaptive computation, recurrent-depth Transformers, and implicit neural layers contribute important pieces of the mechanism, but the survey's scope is narrower. Its common architectural principle is parameter sharing across computational depth: effective depth can increase through repeated application of a learned operator without introducing a distinct parameter block at every step.

Weight sharing alone, however, is not sufficient to define the class studied here. Throughout this survey, recursion serves as the model's internal problem-solving mechanism for a fixed instance. The repeated transition maintains and revises a persistent state that may encode a latent representation, an intermediate answer, a confidence signal, or a collection of candidate solutions. Additional recursive computation can therefore alter the model's prediction, uncertainty, candidate distribution, or halting decision.

This scope distinguishes recursive reasoning from several other forms of iteration. Ordinary recurrent networks primarily reuse computation across input-sequence positions; autoregressive models extend an output sequence one token at a time; and prompting or search-based systems may repeatedly invoke a complete model under an external controller. Although these mechanisms can support reasoning, they pose different architectural questions from an internal parameter-shared loop, particularly with respect to credit assignment, stability, depth extrapolation, halting, and trajectory selection.

Under the terminology adopted in this survey, recursive reasoning models occupy the part of the loop-model landscape in which repeated learned computation directly contributes to constructing, revising, scoring, or selecting a solution for the same problem instance. The following subsection formalizes this boundary and introduces the notation used throughout the survey.

2.3 A Formal Definition of Recursive Reasoning Models

Iteration alone does not define a recursive reasoning model. Autoregressive generation, ordinary sequence-time recurrence, diffusion sampling, repeated prompting, and external search may all involve multiple computational steps. The models studied in this survey share a more specific structure: while the problem instance remains fixed, a persistent internal state is repeatedly updated by a parameter-shared learned transition.

GRAM (Baek et al., 2026b) provides a useful probabilistic starting point for this formalization. Rewritten in the notation of this survey, GRAM begins from a fixed initial state and defines recursive reasoning through a stochastic Markov trajectory,

$$z_t \sim p_\theta(z_t \mid z_{t-1}, e_x), \qquad t = 1, \dots, T, \tag{1}$$

with the conditional prediction obtained by marginalizing over the resulting latent trajectory,

$$p_\theta(y \mid x) = \int p_\theta(y \mid z_{0:T}, x)\, p_\theta(z_{0:T} \mid x)\, \mathrm{d}z_{0:T}. \tag{2}$$

Thus, GRAM defines recursive reasoning as probabilistic latent-trajectory computation: repeated stochastic transitions induce a distribution over possible reasoning paths rather than a single deterministic path. This is a method-specific formulation, with a fixed initialization and explicitly stochastic dynamics.

Although the GRAM readout above is written as depending on the full latent trajectory $z_{0:T}$, the survey-level state used below denotes the complete persistent computation state available to the model. When a method requires trajectory-dependent information at readout, this state can be augmented to retain that information. Formally, one may define $\bar{z}_t = (z_0, \dots, z_t)$, in which case a path-dependent readout $p_\theta(y \mid z_{0:T}, x)$ can be represented as a terminal-state readout $D_\theta(y \mid \bar{z}_T, e_x)$. Thus, the terminal-state decoder in Definition 1 does not require a method's original minimal latent state to contain all readout information.

Building on this GRAM formulation and the broader shared-state characterization of recursive reasoning, we adopt the following survey-level operational definition. It generalizes the same recursive structure to deterministic or stochastic transitions and to fixed, learned, or sampled initial states. The equations describe the broader mechanism of *recursive latent computation*; throughout this survey, we reserve the term *recursive reasoning model* for systems in which this mechanism serves as the learned problem-solving process for a fixed reasoning instance.

Definition 1 (Recursive reasoning model)**.** Let $x \in \mathcal{X}$ be a problem instance and $\mathcal{Y}$ the output space. A recursive reasoning model is a learned system

$$\mathcal{M}_\theta = (E_\theta, \mu_\theta, \mathcal{T}_\theta, D_\theta) \tag{3}$$

that constructs its prediction by repeatedly updating a persistent internal reasoning state.

The input is encoded as

$$e_x = E_\theta(x), \tag{4}$$

and the initial reasoning state is sampled or deterministically initialized according to

$$z_0 \sim \mu_\theta(\cdot \mid e_x). \tag{5}$$

Recursive computation then repeatedly applies the same learned transition kernel,

$$z_{t+1} \sim \mathcal{T}_\theta(\cdot \mid z_t, e_x), \qquad t = 0, \dots, T-1, \tag{6}$$

and the terminal state determines the output distribution

$$p_\theta(y \mid z_T, e_x) = D_\theta(y \mid z_T, e_x). \tag{7}$$

The defining parameter-sharing constraint is that the parameterization of $\mathcal{T}_\theta$ is reused across recursive time. The transition may contain a fixed collection of interacting recurrent submodules, but the number of learned transition parameters is independent of the executed recursion depth T.

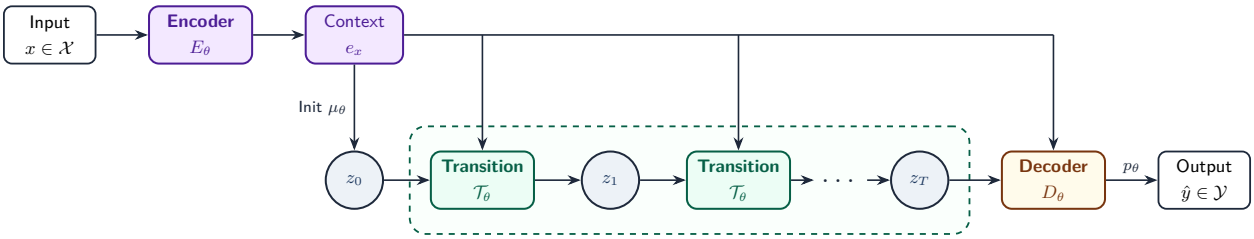

Figure 1: Structural diagram of a recursive reasoning model showing the encoder E_θ, initial state generator μ_θ, parameter-shared transition operator $\mathcal{T}_\theta$, and decoder D_θ.

Figure 1 makes this structural contract explicit: the input is encoded once, the resulting context conditions a parameter-shared recursive transition, and the terminal state is decoded only after the chosen recursion budget has been executed.

The state z_t is intentionally abstract. It may include latent representations, intermediate answer estimates, high- and low-level states, controller or memory variables, confidence scores, retained trajectory information, or a finite population of weighted candidates. Hierarchical and particle-based architectures can therefore be represented by treating their complete persistent computation state as z_t.

For a fixed depth T, Equation 6 induces the trajectory probability measure

$$\mathbb{P}_\theta(\mathrm{d}z_{0:T} \mid x) = \mu_\theta(\mathrm{d}z_0 \mid e_x) \prod_{t=0}^{T-1} \mathcal{T}_\theta(\mathrm{d}z_{t+1} \mid z_t, e_x), \tag{8}$$

and the predictive distribution is obtained by averaging the terminal-state readout over this trajectory law:

$$p_\theta(y \mid x) = \mathbb{E}_{z_{0:T} \sim \mathbb{P}_\theta(\cdot \mid x)} \left[D_\theta(y \mid z_T, e_x) \right]. \tag{9}$$

A final prediction may then be obtained, for example, through

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} p_\theta(y \mid x). \tag{10}$$

Deterministic and stochastic recursion. A deterministic recursive model is the special case in which

$$\mathcal{T}_\theta(\mathrm{d}z' \mid z, e_x) = \delta_{F_\theta(z, e_x)}(\mathrm{d}z'), \tag{11}$$

so that

$$z_{t+1} = F_\theta(z_t, e_x). \tag{12}$$

Given the same input and initialization, such a model follows a fixed trajectory. A stochastic model instead uses a non-degenerate transition kernel and therefore defines a distribution over possible reasoning trajectories. GRAM belongs to the latter class (Baek et al., 2026b).

Fixed and adaptive depth. The recursion depth may be fixed or selected by an optional halting rule. In the adaptive case,

$$\tau = \inf \{ t \geq 0 : H_\theta(z_t, e_x) = 1 \} \wedge T_{\max}, \tag{13}$$

and the output is decoded from z_τ. For stochastic transitions, τ is a stopping time determined by the realized latent trajectory. A learned halting mechanism is therefore an optional controller rather than a defining requirement.

Depth and trajectory width. Recursive depth and trajectory width are distinct inference resources. Depth increases the number of sequential applications of $\mathcal{T}_\theta$. Width samples K trajectories,

$$z_{0:T}^{(k)} \sim \mathbb{P}_\theta(\cdot \mid x), \qquad k = 1, \ldots, K, \tag{14}$$

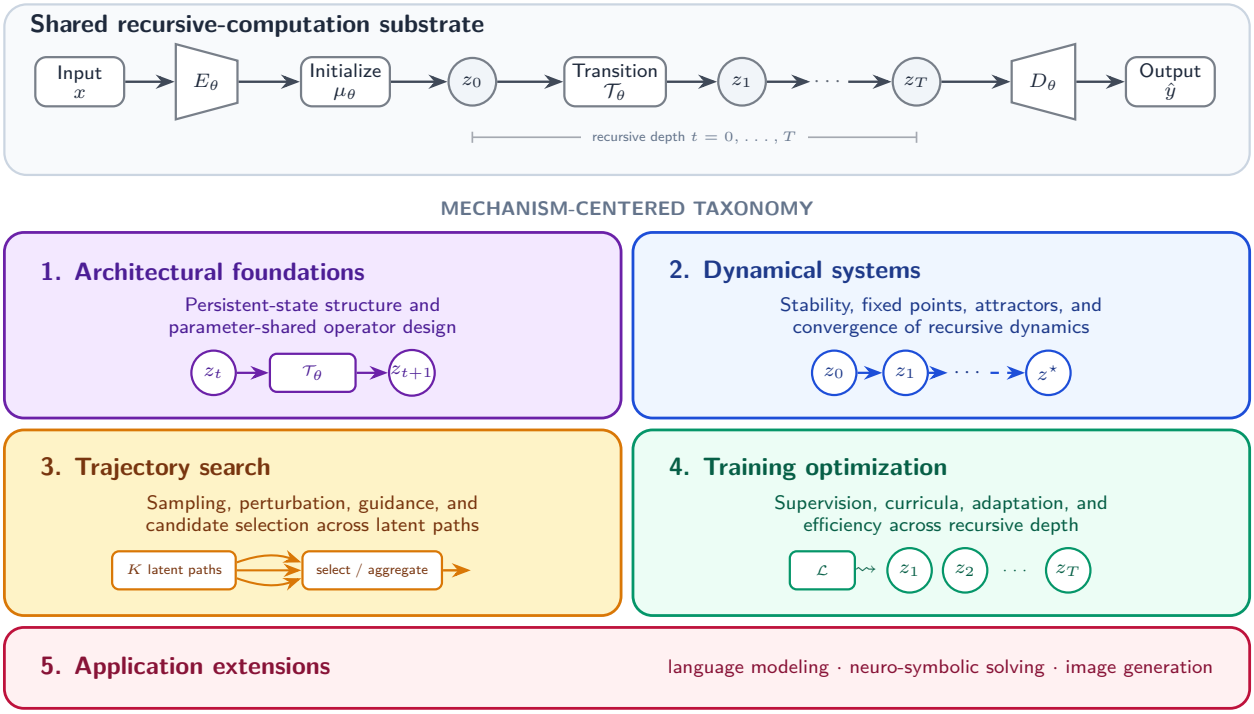

Figure 2: Mechanism-level view of the survey taxonomy. Four mechanism-centered branches examine the shared recursive-computation substrate: Architectural Foundations (state and operator design), Dynamical Systems (stability and convergence), Trajectory Search (optional multi-path exploration and selection or aggregation), and Training Optimization (supervision, curricula, adaptation, and efficiency). Application Extensions transfer these mechanisms to broader systems. $K = 1$ denotes single-trajectory inference and $K > 1$ optional width; input conditioning of $\mathcal{T}_\theta$ is implicit for readability.

and combines their decoded outputs through voting, scoring, verification, or another aggregation rule. In interacting particle methods, the complete particle population and its weights are instead treated as one joint recursive state. Importantly, $K = 1$ denotes single-trajectory inference; it does not imply deterministic recursion. Determinism is determined by the transition kernel, as in Equation 11.

Boundary of the class. Throughout this survey, an RRM satisfies four scope conditions. First, the recursion index denotes internal deliberation time rather than input position or ordinary output-token time. Second, the problem instance remains fixed while the persistent state is revised. Third, the recursively evolved state is answer-coupled: it affects the prediction, confidence, candidate distribution, or halting decision. Fourth, additional deliberation reuses the same transition parameterization rather than introducing a distinct learned block at every step. Consequently, repeated full-model prompting, external tree-search wrappers, ordinary sequence-processing RNNs, and untied depth scaling are not RRMs under the terminology adopted here.

Recursive reasoning models are therefore a structural subclass of latent-reasoning systems: they perform hidden computation specifically through repeated, parameter-shared updates to a persistent state. Figure 1 illustrates this structural contract. Hierarchical loops, adaptive halting, and K parallel trajectories should be understood as optional instantiations rather than universal architectural components.

3 Taxonomy

To organize the rapidly growing literature on recursive latent computation, we use a five-branch taxonomy grounded in the operational contract $\mathcal{M}_\theta = (E_\theta, \mu_\theta, \mathcal{T}_\theta, D_\theta)$ introduced in Definition 1. Four branches

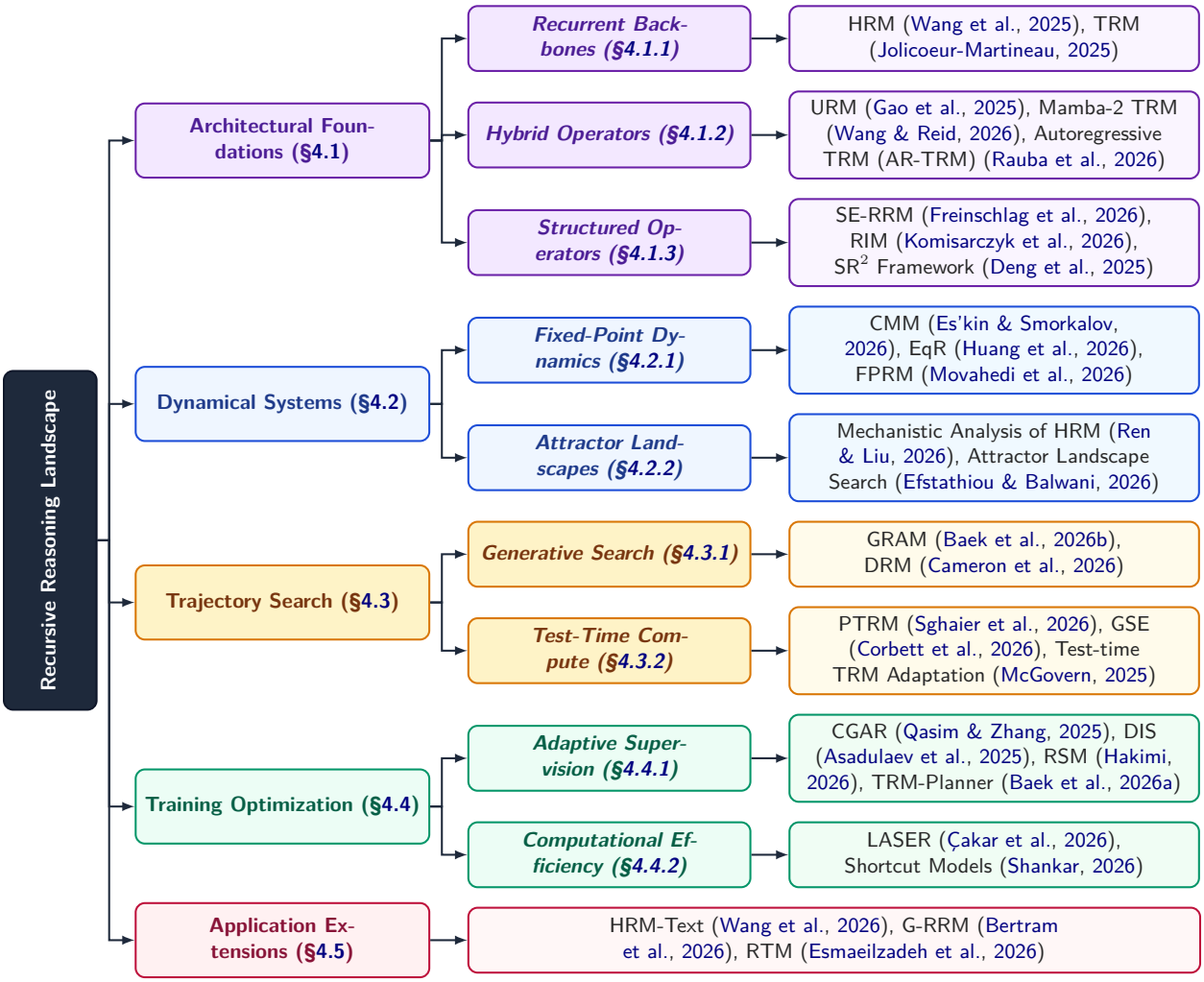

Figure 3: Taxonomy of recursive reasoning models and closely related application extensions.

are mechanism-centered and classify work within the strict RRM scope according to the principal locus of algorithmic innovation. A fifth, *Application Extensions*, tracks work that reuses or adapts closely related recursive latent-computation mechanisms in broader task domains or system interfaces. Inclusion in this fifth branch does not require the complete application to satisfy the strict fixed-instance reasoning criterion.

The taxonomy therefore consists of four functional dimensions and one application-extension branch. Figure 2 provides a mechanism-level view of where the four functional branches intervene in recursive computation and separates Application Extensions as a transfer-oriented branch. Figure 3 then organizes the reviewed methods within these branches.

Architectural Foundations ($\mathcal{Z}$ and $\mathcal{T}_\theta$ Structural Design): Focuses on the topological structure of the persistent latent state $z_t \in \mathcal{Z}$ and the parameterization of the weight-shared transition operator $\mathcal{T}_\theta$. This dimension categorizes approaches into recurrent backbones (Wang et al., 2025; Jolicoeur-Martineau, 2025), hybrid attention/SSM operators (Gao et al., 2025; Wang & Reid, 2026), and structured operators (Freinschlag et al., 2026; Komisarczyk et al., 2026; Deng et al., 2025).

Dynamical Systems ($\mathcal{T}_\theta$ Asymptotics and Trajectories): Analyzes recursive execution as a discrete or continuous dynamical system $z_{t+1} \sim \mathcal{T}_\theta(\cdot \mid z_t, e_x)$. This dimension encompasses fixed-point equilibrium models (Es'kin & Smorkalov, 2026; Huang et al., 2026; Movahedi et al., 2026) as well as mechanistic investi-

gations into attractor landscapes and latent convergence stability (Ren & Liu, 2026; Efstathiou & Balwani, 2026).

Trajectory Search ($\mathbb{P}_\theta(\cdot \mid x)$ Exploration): Extends single-path deterministic recursion to stochastic latent trajectory distributions, enabling multi-hypothesis latent exploration and test-time compute allocation. This dimension covers generative search formulations (Baek et al., 2026b; Cameron et al., 2026) and inference-time stochastic particle guidance mechanisms (Sghaier et al., 2026; Corbett et al., 2026).

Training Optimization and Adaptation ($\mathcal{M}_\theta$ Objectives and Efficiency): Addresses the backpropagation and credit assignment challenges across extended recursion depths T. This dimension includes adaptive intermediate supervision curricula (Qasim & Zhang, 2025; Asadulaev et al., 2025; Hakimi, 2026) alongside architectural strategies for training and inference efficiency (Çakar et al., 2026; Shankar, 2026).

Application Extensions (Domain and System Integration): Collects work whose principal contribution is not a new recursive reasoning mechanism, but the reuse or adaptation of closely related recursive latent-computation mechanisms in a substantially different setting. HRM-Text adapts hierarchical recurrence to language-model pretraining (Wang et al., 2026); G-RRM uses an SE-RRM to guide exact symbolic solvers (Bertram et al., 2026); and RTM transfers TRM-style recursive refinement to the latent mapping network of image generators (Esmaeilzadeh et al., 2026). These papers can inherit mechanisms from the first four branches, but grouping them by application avoids mischaracterizing domain transfer as a new architectural or search principle.

4 Review of Key Papers

This section synthesizes the key contributions through the taxonomy developed in this paper. Rather than presenting the literature as a sequence of isolated proposals, it traces how recursive reasoning has evolved across architectural design, dynamical interpretation, trajectory-level search, and training and efficiency optimization, before examining application-oriented extensions that transfer recursive latent-computation mechanisms to language modeling, neuro-symbolic solving, and image generation.

4.1 Architectural Foundations

Architectural work defines what information is preserved across recursive steps and what transformation is repeatedly applied to it. The literature progresses from recurrent backbones that establish latent iterative refinement, to hybrid operators that redesign the shared transition, and finally to structured operators that embed stronger inductive biases into the recursive process.

4.1.1 Recurrent Backbones

The modern recursive-reasoning literature begins with the Hierarchical Reasoning Model (HRM) (Wang et al., 2025), which reframed structured problem solving as repeated latent computation rather than explicit chain-of-thought generation. HRM maintains a rapidly updated low-level state z_L and a more slowly updated high-level state z_H. Given the input representation

$$\tilde{x} = f_I(x; \theta_I), \tag{15}$$

HRM executes N high-level cycles, each containing T low-level updates. Using c to index high-level cycles and j to index low-level updates within a cycle, the dynamics can be written as

$$\begin{aligned} z_L^{c,j} &= f_L\left(z_L^{c,j-1}, z_H^{c-1}, \tilde{x}; \theta_L\right), \qquad j = 1, \ldots, T, \\ z_H^{c} &= f_H\left(z_H^{c-1}, z_L^{c,T}; \theta_H\right), \qquad c = 1, \ldots, N. \end{aligned} \tag{16}$$

Here, $z_L^{1,0} = z_L^0$, and for $c > 1$ the low-level state is carried between cycles according to $z_L^{c,0} = z_L^{c-1,T}$. After N complete cycles, the output is decoded from the final high-level state:

$$\hat{y} = f_O\left(z_H^N; \theta_O\right). \tag{17}$$

Although the equations write the module inputs as separate arguments, the reported implementation combines the multiple inputs to each recurrent module using element-wise addition before applying the corresponding Transformer block.

The slower state is intended to organize abstract planning, while the faster state performs local refinement. Training proceeds through deeply supervised segments: recurrent states are carried forward but detached, and gradients are retained only through the final updates of each segment. HRM motivates this approximation through an implicit fixed-point argument and combines it with an ACT-style controller that predicts whether another segment is required. These choices reduce the memory cost of full backpropagation through the latent trajectory, but they also entangle the architecture with its optimization procedure. Two recurrent modules, a nested update schedule, deep supervision, learned halting, and truncated credit assignment are introduced together. The reported results of the compact 27M-parameter model on Sudoku, maze, and ARC-AGI tasks established latent recursion as a credible alternative to parameter and token scaling, yet the bundled design left open whether hierarchy itself, rather than repeated refinement, augmentation, or the broader training recipe, was responsible for the gains. HRM therefore served less as a settled account of reasoning than as the hypothesis that subsequent work would simplify and test.

TRM (Jolicoeur-Martineau, 2025) sharpened that test by removing most of the hierarchy. A single small network refines an auxiliary state and a solution state through nested cycles,

$$z_t^{(r)} = f_\theta\Big(x + y_t + z_t^{(r-1)}\Big), \qquad y_{t+1} = f_\theta\Big(y_t + z_t^{(n)}\Big), \tag{18}$$

while state is again carried between supervision segments with detached gradients. This simplification was methodologically important: a 7M-parameter model could match or exceed the more elaborate HRM on several benchmarks, making shared recursive refinement the natural baseline for later work. It also made the scientific question more precise. Because one two-layer core performs both latent and answer updates, any functional specialization must emerge from the state inputs and update schedule rather than separate parameter sets. Because most inner updates are executed without gradients, the retained transition must learn to act on states produced by the model's own detached computation.

The simpler scaffold did not, however, remove the influence of deep supervision, task embeddings, augmentation, halting calibration, or repeated inference. Later controlled evaluations made this limitation explicit. On ARC-AGI-1, Roye-Azar et al. (2025) found that much of TRM's reported performance depends on large test-time ensembles, augmentation, and puzzle-identity conditioning; recursion was often effectively shallow, and identity-transformation performance collapsed under controls. This does not eliminate the value of the architecture, but it changes the interpretation of that value. A small recursive model may be useful because it generates many inexpensive candidate solutions, even when extending one trajectory produces little additional benefit. Nominal effective depth should therefore not be treated as evidence of deep computation without intermediate-prediction analysis, perturbation tests, and results at realistic sampling budgets.

Figure 4 summarizes the architectural distinction between HRM's nested two-timescale recurrence and TRM's shared single-core refinement schedule.

Mechanistic work subsequently shifted the debate from architectural labels to information flow. Using activation patching and related diagnostics, Miyanishi & Morimura (2026) show that interaction locality depends on task geometry, transition channel, perturbation scale, and the recursive time edge; it does not follow a simple division in which the low-level state is local and the high-level state is global. In released HRM and TRM checkpoints, high-level within-state interactions can be at least as local as low-level interactions, while cross-cycle propagation spreads locally aggregated information over time. A complementary controlled study by Shen et al. (2026) asks whether shared parameters can nevertheless support distinct functions. Under Asymmetric Input Recurrence, the same recurrent Transformer develops a proposal-like state that commits earlier and a revision-like state that retains uncertainty, but the specialization weakens when the update type is not identifiable.

Together, these studies replace the biological hierarchy narrative with a narrower and better-supported claim: functional roles can emerge under weight sharing, but they are induced by update schedules, input asymmetry, and task structure rather than guaranteed by naming states "high" and "low." They also

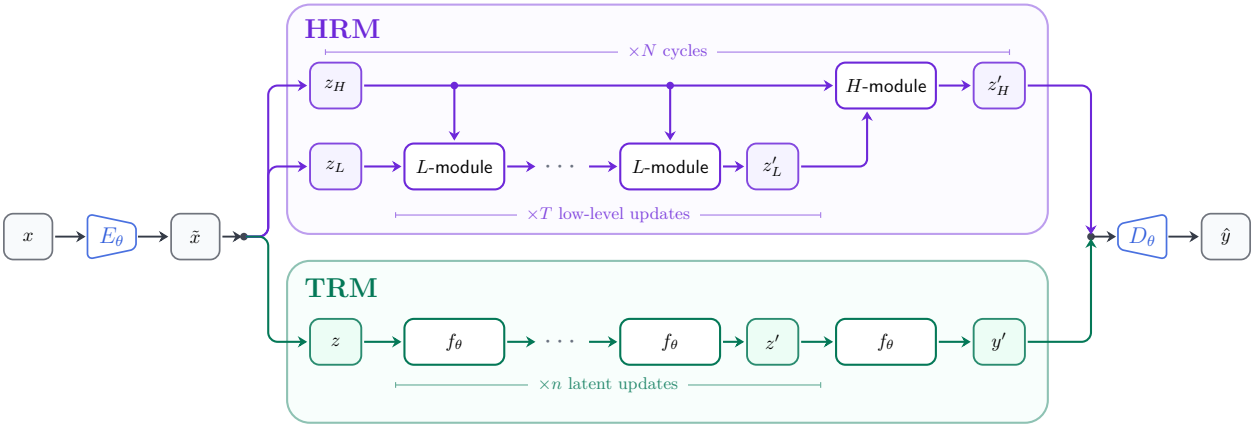

Figure 4: Schematic comparison of the recurrent update schedules in HRM and TRM. HRM alternates T fast low-level updates with a slower high-level update across N cycles, whereas TRM reuses a single shared core f_θ for repeated latent-state updates followed by solution-state refinement. Shared input encoder E_θ and output decoder D_θ are shown only to emphasize the contrast in the recursive core.

show why attention patterns alone are inadequate evidence. The locality study relies primarily on finite-noise causal patching, whereas the specialization study uses state freezing and architectural ablations to test whether the inferred roles survive intervention. Both remain limited to small models and structured domains, and neither yet turns its mechanistic findings into a stronger training objective. Their conclusions are therefore best treated as well-motivated hypotheses about recursive computation rather than universal laws.

Across HRM, TRM, and their mechanistic follow-ups, the evidence supports a restrained conclusion. Reusing a compact transition over latent time is a powerful computational primitive, but neither a two-timescale architecture nor a long nominal unroll establishes that a model performs globally organized, step-by-step reasoning. The central question is no longer whether recurrence can work, but which states, asymmetries, supervision signals, and inference controls make additional computation reliably useful. Extensions that transplant these recurrent backbones into substantially different application settings are discussed separately in Section 4.5.

4.1.2 Hybrid Operators

After HRM and TRM established parameter-shared refinement, the architectural question shifted from whether to recur to what should recur. URM (Gao et al., 2025) revisits Universal Transformer recurrence (Dehghani et al., 2019), strengthening the shared block with short convolutional mixing and ConvSwiGLU-style nonlinear processing. Its ARC-AGI results indicate that weight tying is not sufficient by itself: the repeated operator must also provide enough local mixing and nonlinear capacity to transform an evolving grid representation. The Mamba-2 attention hybrid studied by Wang & Reid (2026) reaches a complementary conclusion. Replacing TRM's Transformer block with a parameter-matched state-space/attention module leaves pass@1 close to the original model, while producing clearer gains at larger pass@k. The improvement therefore appears more strongly in candidate coverage than in the top-ranked single trajectory, suggesting that operator design can reshape the distribution of solutions without resolving the separate problem of selection. The evidence is also benchmark-specific: URM is evaluated primarily on ARC-style transformations, whereas the Mamba hybrid is judged partly through multi-sample performance. Broader conclusions require matched studies across algorithmic, spatial, and symbolic tasks under the same depth and sampling budgets. Together, these results establish convolution-enhanced and state-space/attention hybrids as viable recurrent transitions, but not as universally superior replacements for attention.

The prediction interface places an important boundary on this conclusion. Rauba et al. (2026) hold the block design, causal token stream, next-token objective, and block-pass budget fixed while moving from untied depth to recurrent depth, two-stream recurrence, and a full Autoregressive TRM (AR-TRM). Flat two-level variants can remain competitive, but the nested model provides no reliable gain and often degrades sharply on character-level copying, reversal, and addition. Recurrence that supports parallel refinement of a fixed structured output therefore does not automatically transfer to a decoder that must commit one token at a time. Hybrid-operator research is best understood as a search over transition geometry and compute placement whose success depends jointly on the block family, task structure, and output interface.

4.1.3 Structured Operators

A second architectural branch gives the recurrent operator explicit knowledge about the structure of the solution space. SE-RRM (Freinschlag et al., 2026) begins from the observation that Sudoku digits and many ARC colors are labels rather than intrinsically ordered quantities. Earlier RRMs can learn this symmetry through extensive relabeling augmentation, whereas SE-RRM builds symbol-permutation equivariance into the architecture by introducing an explicit symbol axis and applying position-wise and symbol-wise attention in an axial pattern (Ho et al., 2019). This makes a consistent relabeling of the input induce the same relabeling of the output. The resulting model reports stronger robustness on standard Sudoku, extrapolation from 9×9 training to smaller and larger grids, and competitive ARC-AGI performance with fewer parameters and less augmentation. The broader lesson is that recurrence becomes more sample-efficient when each update respects a known invariance, although such benefits depend on identifying the correct symmetry group in advance.

RIM (Komisarczyk et al., 2026) introduces structure at the level of the inference algorithm rather than the representation. It interprets recursive neural reasoning as repeated proposal, solution update, and reweighting, drawing inspiration from Sequential Monte Carlo (Doucet et al., 2001). Within this framework, TRM appears as a special case, while an explicit learned reweighter can prioritize more promising hypotheses and improve performance on ARC-AGI, Sudoku, and tabular tasks. The analogy is useful because it exposes operations that many RRMs perform only implicitly: constructing a candidate, transforming it, and estimating its value. It should not, however, be confused with a guarantee inherited from classical SMC; the learned scores still require calibration, and their reliability can change when the candidate distribution shifts.

SR^2 (Deng et al., 2025) reaches a related destination from a causal-selection perspective. A single shared Transformer alternates between reflection, in which the latent state is repeatedly updated with access to the input, and self-refinement, in which input injection is removed so that dependencies within the internal representation must be resolved. Periodic intermediate alignment supplies an additional training signal. The reported results on Sudoku and maze tasks show that scheduling when the model may consult the observation can substitute for a physically separate hierarchy, and ablations indicate that both phases matter. Yet its causal interpretation and convergence assumptions remain primarily conceptual, and repeated application still carries substantial compute despite the small parameter count. Taken together, SE-RRM, RIM, and SR^2 show three complementary ways to constrain recursion: encode a task symmetry, expose an inference-like proposal-and-weighting decomposition, or control the flow of observational information. These choices can make the loop more interpretable and efficient, but they also trade architectural generality for assumptions about the problem's latent structure.

4.2 Dynamical Systems

Beyond their architectural description, recursive models can be studied as dynamical systems whose hidden states evolve under repeated application of a shared transition. This perspective shifts the analysis toward convergence, stability, and the geometry of the successful and unsuccessful states reached during inference.

4.2.1 Fixed-Point Dynamics

As recursive depth increased, the literature began to treat the loop not only as an architecture but as a dynamical system whose trajectories should remain stable and terminate for principled reasons. Es'kin & Smorkalov (2026) take the most explicit continuous-time route. Their Contraction Mapping Model recasts

discrete recursion through Neural ODE and SDE formulations and augments task loss with equilibrium- and Jacobian-oriented stability terms, together with hyperspherical repulsion intended to prevent feature collapse. The reported Sudoku and maze results, including strong accuracy at very small parameter counts, suggest that stability regularization can improve both robustness and compression. Nevertheless, contraction is also a restrictive inductive bias: forcing trajectories toward stable equilibria can suppress oscillatory or branching behavior that may be useful for search, and the empirical guarantees remain tied to the assumptions and benchmarks studied in the paper.

EqR (Huang et al., 2026) develops the equilibrium interpretation into a test-time scaling principle. Rather than assuming that more iterations are beneficial by themselves, it aims to shape task-conditioned attractor landscapes so that valid solutions correspond to stable fixed points. Accuracy is then analyzed jointly with convergence residuals and with two compute axes: deeper evolution of one state and broader exploration from multiple initializations or perturbations. The central empirical message is that extreme depth is useful only when the learned dynamics remain solution-aligned; convergence toward the wrong attractor merely makes an error more stable. Breadth therefore complements depth by increasing the probability of entering a productive basin.

FPRM (Movahedi et al., 2026) addresses the same problem through a more conventional looped Transformer design. It argues that post-normalized HRM/TRM blocks keep recurrent activations bounded but can impair signal propagation at large effective depth, while naive pre-normalization improves propagation at the cost of growing activation norms. FPRM combines pre-normalization, learnable residual scaling, repeated input injection, and damped updates. It then uses the relative fixed-point residual as an intrinsic stopping signal, reducing the damping step when progress stalls rather than relying solely on a separately trained halting head. The resulting 7M-parameter model reports competitive results across Sudoku, maze, state tracking, and ARC-AGI and adapts its computation to instance difficulty.

These works converge on a common design criterion: an RRM should expose whether further updates are moving the state toward a stable, answer-bearing region. They differ in how strongly this criterion is imposed. CMM regularizes the vector field toward contraction, EqR organizes and explores an attractor landscape, and FPRM combines signal-propagation engineering with residual-based halting. None provides a universal guarantee. Bounded states need not converge, local Jacobian conditions need not describe distant trajectories, and a small residual does not certify that the decoded answer is correct. Fixed-point diagnostics are therefore valuable control signals, but they remain most reliable when paired with task verifiers or empirical calibration.

4.2.2 Attractor Landscapes

Mechanistic work on attractor landscapes emerged from an apparent contradiction: recursive models often improve with additional computation, yet their predictions do not always become steadily more accurate. Ren & Liu (2026) examine this tension in HRM. They find that the fixed-point premise used to motivate its approximate gradient can fail in practice: the model may overwrite a correct assignment and can fail on Sudoku instances with only one unknown cell. Across supervision segments, loss frequently stays on a plateau and then drops abruptly, producing a grokking-like transition rather than smooth refinement. Projected latent trajectories and conflict-based analyses reveal both correct and spurious fixed points, suggesting that the model often commits to an attractor early and remains trapped when that attractor encodes an invalid solution.

This diagnosis leads directly to Augmented HRM. Data mixing across puzzle difficulties improves the stability of already solved states, while equivalent input relabelings and checkpoints sampled from later training stages create distinct deterministic trajectories at inference. Majority voting over successfully halted outputs converts this diversity into a width-scaling mechanism. The combined procedure raises the paper's Sudoku-Extreme accuracy from roughly 55% to 96.9% and also substantially improves Maze-Hard. The result is compelling because the intervention follows from the mechanistic account: if one trajectory makes an early basin choice, then perturbing inputs and model parameters supplies additional chances to enter a better basin. Its cost is equally clear. The strongest numbers require many passes and exploit task-specific symmetries; checkpoint bootstrapping also stores and evaluates several late-training model states rather

than improving one deployed trajectory. The aggregation procedure therefore combines three sources of diversity (training examples, equivalent inputs, and parameter realizations) whose individual contributions and compute costs should be reported separately. Moreover, low-dimensional projections and heuristic conflict landscapes cannot establish a complete causal description of HRM's internal algorithm.

Efstathiou & Balwani (2026) report a closely related picture for the MLP version of TRM on Sudoku-Extreme. Sparse autoencoders, PCA trajectories, and loss traces indicate that computation is front-loaded: a large fraction of solved instances are resolved within the first few supervision steps, after which returns diminish and dominant features can settle into approximately periodic patterns. Successful and failed trajectories diverge early; the former continue toward low-loss regions, whereas the latter stabilize in high-loss states interpreted as spurious attractors. This motivates a simple multi-start intervention: perturb the initial answer state with Gaussian noise, run several short trajectories, and select the candidate with the highest ACT confidence. With a substantial number of starts, the paper reports accuracy near 98% on its analyzed Sudoku subset without retraining the base model. As in PTRM-style inference, however, the procedure delegates final choice to a confidence head trained under the original deterministic regime. Its success therefore depends on both basin coverage and the ranking quality of ACT confidence after the initialization distribution has been changed.

Together, the two studies change the meaning of test-time scaling. Additional depth is not uniformly valuable once a trajectory has entered a stable basin; at that point, width can be more useful because it changes the initial conditions or model realization from which the basin is reached. The studies also differ in where diversity enters. Augmented HRM uses equivalent observations and late-training checkpoints, whereas the TRM intervention perturbs the initial latent answer while holding the checkpoint fixed. Comparing these interventions under equal forward-pass budgets would help distinguish whether robustness comes primarily from model diversity, input symmetry, or local exploration of one learned landscape. They also expose a recurring vulnerability: the same halting or confidence head used to select trajectories may be miscalibrated under perturbation, and an attractor can be dynamically stable while decoding to an invalid answer. The attractor account is therefore a productive organizing hypothesis, one that explains abrupt improvement, early divergence, diminishing depth returns, and the success of multi-start inference, but its present evidence is concentrated on Sudoku and specific checkpoints. Establishing whether similar landscapes govern language, planning, or open-ended reasoning will require causal interventions at larger scale and evaluations in which stable outputs are not automatically verifiable.

4.3 Trajectory Search

Because a single recursive trajectory may converge prematurely or enter an unproductive region of latent space, several methods broaden computation beyond deterministic refinement. These approaches allocate additional test-time or training-time resources to generating, adapting, or selecting among alternative reasoning trajectories.

4.3.1 Generative Search

The trajectory-search literature extends recurrence from the refinement of one deterministic hypothesis to the construction of a distribution over possible computations. GRAM (Baek et al., 2026b) makes this shift part of the learned model. Its transition samples a state-dependent perturbation around a deterministic update,

$$z_{t+1} \sim p_\theta(z_{t+1} \mid z_t, e_x)\,, \tag{19}$$

so repeated recursion induces multiple latent trajectories rather than copies of a single path. A target-conditioned posterior is used during amortized variational training, while inference samples from the learned prior. GRAM can consequently scale along both depth and width, select trajectories by voting or a learned latent process reward model, recover multiple valid solutions in constraint tasks, and operate as an unconditional generative model when the conditioning input is absent. This is a broad conceptual unification of reasoning, uncertainty, and generation, but it requires end-to-end stochastic training and a truncated surrogate to the full trajectory ELBO, making it more complex than inference-only perturbation methods.

Cameron et al. (2026) approach generative recursion through the training path rather than a probabilistic transition model. Denoising Recursion Models corrupt a target and train one shared operator to undo that corruption over several recursive steps, thereby supplying a curriculum of intermediate states while matching the multi-step behavior used at test time. The associated State Perturbation Recursion Model instead perturbs the model's incumbent latent state during training, forcing the transition to recover from neighborhoods around its own trajectories. The reported ARC-AGI experiments favor DRM in data-rich regimes, while SPRM can help as a regularizer in smaller-data settings. The larger contribution is the non-greedy objective: each update is trained as part of a multi-step recovery process rather than as an isolated jump directly to the answer. Its behavior still depends on the corruption process, noise schedule, recursive window, and remasking policy.

Taken together, GRAM and DRM/SPRM broaden recursive reasoning beyond greedy deterministic refinement in two different ways: GRAM learns a stochastic distribution over latent trajectories, whereas DRM and SPRM train a shared transition to recover from deliberately perturbed states. Both therefore change the trajectory-generation mechanism itself. Application-oriented uses of recursive refinement in continuous image generation are separated into Section 4.5, where the central contribution is domain transfer rather than search over reasoning trajectories.

4.3.2 Test-Time Compute

The first test-time scaling strategy broadens inference over multiple latent trajectories. PTRM (Sghaier et al., 2026) starts from a basic limitation of deterministic TRM: rerunning the same checkpoint from the same state reproduces the same path. It injects Gaussian noise at every deep-recursion step, executes K independent rollouts, and selects a terminal answer with the model's existing Q-head. This creates a width axis without retraining or task-specific input transformations. Reported gains, including 87.4% to 98.75% on Sudoku-Extreme and large improvements on Pencil Puzzle Bench, show that some failures arise from entering the wrong basin rather than lacking representational capacity. At the same time, the gains expose a second bottleneck: width is useful only when the selector ranks rare correct trajectories above more common failures. In the authors' validation analysis, Q-selection nearly matches the pass@K oracle, whereas majority voting improves much less because correct answers can remain a small minority. Parallel rollouts reduce sequential latency relative to deeper single trajectories, but total accelerator use and energy still scale with K, and selector calibration cannot be assumed to transfer across tasks, checkpoints, depths, or perturbation scales.

GSE (Corbett et al., 2026) develops the same idea into a more explicit approximate-inference framework. A frozen TRM becomes the zero-noise, single-particle limit of a stochastic particle system: Gaussian perturbations propose neighboring trajectories, while the Q-head guides the particle cloud toward promising regions. The main addition is diagnostic rather than merely algorithmic. Local stability, guide alignment, and cloud-token entropy provide label-free estimates of whether exploration is likely to uncover new solution basins or simply diversify errors. Sudoku results illustrate the favorable regime; maze experiments show that a misaligned guide can make additional particles unproductive. GSE thus reframes the question from whether sampling helps in the abstract to whether the transition, perturbation process, and internal guide are jointly suitable for sampling. Its particle view is more informative than best-of-K alone, although the guide is still inherited from a halting or correctness objective rather than trained explicitly for trajectory selection.

A different test-time intervention changes model parameters rather than multiplying latent rollouts. In the adaptation setting, McGovern (2025) pretrains TRM on public ARC data and continues optimization for previously unseen tasks under competition-style limits. Full fine-tuning outperforms updates restricted to task embeddings or low-rank adapters, but the semi-private score remains modest and sensitive to both the pretraining distribution and adaptation budget. Each task provides only a few demonstrations, yet thousands of gradient steps must be allocated without access to hidden test outputs. The study therefore shows that a recursive backbone can be specialized from a shared prior, while also demonstrating that recurrence alone does not solve few-example adaptation; the update rule must become faster, more transferable, and less brittle.

Viewed together, these methods show that "more test-time compute" already names distinct interventions even within this branch. PTRM and GSE allocate compute to parallel latent exploration, whereas test-time adaptation allocates it to parameter updates for a new task. Their failure modes are correspondingly different: selector miscalibration, guide misalignment, or adaptation overfitting. Meaningful comparison therefore requires more than an accuracy number: studies should report trajectory count, sequential depth, adaptation steps, selection rule, neural inference cost, and wall-clock latency. Neuro-symbolic solver guidance is treated separately as an application-level integration in Section 4.5.

4.4 Training Optimization

The value of recursive computation depends not only on the transition operator, but also on how that operator is supervised and how efficiently repeated updates can be executed. The methods in this family therefore improve the learning signal across recursive steps or reduce the memory and latency costs associated with long computational trajectories.

4.4.1 Adaptive Supervision

Training methods in this family address a common mismatch: a shared transition is executed many times, yet ordinary supervision does not guarantee that each execution contributes useful computation. The first response is to vary how much recursion the model sees during optimization. CGAR (Qasim & Zhang, 2025) uses a Progressive Depth Curriculum that moves from shallow to medium and then full recursion, together with Hierarchical Supervision Weighting that exponentially reduces the contribution of later supervision segments in response to observed gradient decay. On Sudoku-Extreme, the reported combination lowers wall-clock training from 10.93 to 6.38 hours with only a small accuracy change, and ablations identify the depth curriculum as the main source of the speedup. The result turns recursion depth into a schedulable training variable rather than a fixed architectural constant, although the best schedule and weighting are likely to remain task- and implementation-dependent.

A second line of work asks what, if anything, intermediate iterations should predict. DIS (Asadulaev et al., 2025) argues from a policy-improvement view that nominal depth becomes "dead compute" when successive updates fail to increase the advantage of the correct answer. Deep Improvement Supervision therefore constructs a monotone sequence of partially corrupted targets, giving each recursive step a concrete sub-goal closer to the final output. In its TRM testbed, the method removes the learned halting module, reduces the number of supervision and latent steps, lowers forward-pass count by roughly 18×, and remains competitive on ARC with a very small model. Its strength is also its main assumption: the method requires an informative improvement path. Simple corruption is broadly applicable, but branching or non-monotone reasoning may not admit a natural sequence of progressively corrected targets.

RSM (Hakimi, 2026) takes the contrasting position that direct intermediate losses can make the transition too dependent on a particular unroll. It fully detaches recurrent history, treats early iterations as warm-up, and applies loss only at the final step, while allowing inner and outer depth to grow independently and applying stochastic depth to outer transitions. The aim is a depth-agnostic update map that reliably settles rather than a trajectory trained to match the answer at every iteration. The paper reports more than 20× faster training than its TRM comparison, strong Sudoku and maze performance, and extrapolation from tens of training iterations to thousands at inference. Settling behavior can also act as a warning signal when a trajectory fails to converge. These results are promising, but stable convergence is not equivalent to correctness, and final-step-only gradients still provide limited direct credit for long-range dependencies.

TRM-Planner (Baek et al., 2026a) locates the additional supervision outside the student's online trajectory. A frozen teacher is unrolled over multiple depths and stochastic rollouts while its learned stopping decision is ignored. A label-aware planner scores candidate states, can search over stop/continue decisions, and caches selected tokens, logits, step indices, and quality metadata. The student then trains with the ordinary TRM objectives plus hard or StableMax-space soft distillation, leaving deployment-time inference unchanged. With 7M parameters, the paper reports pass@2 gains from 43.1% to 48.1% on ARC-AGI-1 and from 6.7% to 9.2% on ARC-AGI-2, together with a larger improvement in its reproduced Sudoku setting. Controls indicate that cached rollout distillation already explains most of the ARC-AGI-1 gain, so label-aware step selection

supplies a smaller additional contribution. The method also requires labels during cache construction and exchanges offline rollout and storage cost for reusable student supervision.

These approaches differ less in whether recursion should be supervised than in where the learning signal should enter. CGAR schedules the amount and weighting of recurrent computation; DIS supplies an explicit improving path; RSM withholds intermediate targets to shape a reusable transition; and TRM-Planner distills selected teacher states offline. No single contract dominates across the reported settings. The appropriate design depends on whether the main bottleneck is wall-clock cost, unproductive intermediate updates, depth extrapolation, or sensitivity to halting and target depth. Future comparisons should therefore separate compute savings from accuracy changes and report gradient horizon, number and type of supervised states, offline teacher cost, and the inference depths over which the learned transition remains stable.

4.4.2 Computational Efficiency

Computational efficiency is an important consideration for recursive reasoning models because their repeated state updates introduce costs at both training and inference time. Recent work attacks three complementary bottlenecks. LASER (Çakar et al., 2026) reduces activation memory during training, Shortcut Models (Shankar, 2026) reduce the number of recursive steps executed during inference, and quantization work studies how low-precision arithmetic perturbs a transition that is reused many times. Table 7 in Section 5.4 summarizes the first two methods, for which the papers report compact accuracy–efficiency trade-offs. The quantization result is discussed separately below because its central variable is numerical format and scaling granularity rather than a directly comparable memory or FLOP ratio.

LASER exploits the observation that activations produced across recursive iterations often exhibit substantial low-dimensional structure. Instead of retaining every full activation required for backpropagation, the method represents selected activations using an adaptively maintained low-rank basis. On an 11×11 maze-pathfinding TRM with 24 recursive steps, the $k = 128$ configuration reduces measured activation memory from 2.85 GB to 1.14 GB (about 60%) while validation token accuracy changes from $97.82 \pm 2.78\%$ to $98.46 \pm 2.01\%$ over five seeds (Çakar et al., 2026). The exact maze-solve rate has high variance in this small validation setup, so the evidence is strongest for memory reduction without a statistically significant degradation in validation accuracy, not for a general accuracy improvement. The result is also task-specific: the paper evaluates a single structured maze regime whose activation subspace may be unusually compressible.

Shortcut Models address a different source of cost: the sequential latency caused by repeatedly applying the recursive transition function. A student is trained to approximate multi-step latent progress with learned jumps, with a repair mechanism used to bring jumped states back toward the teacher trajectory. On ARC-AGI-1, the reported 8-, 4-, and 2-hop variants use roughly $0.125\times$, $0.25\times$, and $0.5\times$ the baseline TRM inference FLOPs and obtain 28.6%, 34.2%, and 39.7%, respectively, compared with 40.4% for the TRM baseline; a 1-hop model at the full baseline budget reports 40.8% (Shankar, 2026). The result therefore exposes an explicit compute–accuracy curve rather than a single efficiency number, and it also adds teacher-distillation and repair-model training overhead that should be reported separately from deployment FLOPs.

A complementary deployment question is numerical precision. (Ingolfsson et al., 2026) show that quantization error in an RRM cannot be treated as a one-pass perturbation because the same quantized transition is reused across many recursive steps. In their experiments, 8-bit quantization preserves accuracy, whereas naive per-tensor 4-bit activation quantization introduces a systematic bias that accumulates along the trajectory and can collapse exact Sudoku accuracy. Blockwise scaling restores the transition much more effectively, including on deeper EqR-style recursion. This result adds a recursion-specific efficiency constraint: compression should be evaluated not only by one-step approximation error, but also by how numerical bias evolves under repeated application.

Together, these studies show that the computational overhead of recursive reasoning is not confined to a single stage. LASER reduces the memory cost of retaining intermediate states during training, Shortcut Models reduce the temporal cost of traversing those states during inference, and quantization studies expose the numerical stability cost of repeatedly reusing an approximate transition. These results also highlight that efficiency should not be measured solely by the number of recursive steps, the dimensionality of stored activations, or nominal bit width. A complete evaluation should additionally account for achieved accuracy,

actual memory or latency reduction, added parameters, training overhead, and whether the efficiency gains remain consistent across tasks.

4.5 Application Extensions

The papers in this branch reuse or adapt recursive latent-computation mechanisms closely connected to RRMs, but their complete systems need not all satisfy the strict fixed-instance definition adopted in Section 2.3. We separate them from the four mechanism-centered branches because their primary contributions concern deployment in substantially different application settings: HRM-Text targets language-model pretraining, G-RRM integrates recursive neural computation with exact symbolic solvers, and RTM adapts recursive refinement to continuous image generation. Grouping them as application extensions records the broader transfer of the recursive-computation paradigm without classifying every such system as a strict RRM.

HRM-Text (Wang et al., 2026) is the clearest extension from structured fixed-output puzzles to autoregressive language modeling. The model keeps HRM's dual-timescale recurrent organization, using two high-level cycles with three fast low-level updates inside each cycle, but couples this backbone to two stabilization and training changes that are specific to language-scale recurrence. MagicNorm caps each recurrent module with normalization to control forward activation growth, while a warmup deep-credit-assignment schedule initially backpropagates through only the final two recurrent steps and gradually expands the gradient horizon to five. The pretraining objective also departs from ordinary raw-text next-token training: HRM-Text is trained from scratch on instruction–response pairs, applies negative log-likelihood only to the response, and uses PrefixLM masking so instruction tokens can attend bidirectionally while response generation remains causal. Under the paper's matched-training-FLOPs study, the 1B HRM model outperforms the compared 1B Looped Transformer and RINS baselines across the reported benchmark suite and is competitive with larger Transformers. The final 1B checkpoint, trained on 40B unique tokens, reports 60.7 on MMLU, 81.9 on ARC-C, 82.2 on DROP, 84.5 on GSM8K, and 56.2 on MATH, while the authors estimate substantially lower training-token and compute budgets than the open models used for comparison. These results show that hierarchical recurrence can be made viable at language-model scale, but they should not be interpreted as an isolated test of recurrence: the ablations show material gains from the task-completion objective and PrefixLM as well as from the architecture. Moreover, HRM-Text does not use adaptive computation time, so its fixed recurrent schedule still adds inference-time computation relative to a single-pass Transformer.

G-RRM (Bertram et al., 2026) uses an SE-RRM as a probabilistic prior for exact symbolic search. Given a Sudoku instance, the neural model scores variable–value assignments and uses those scores to order candidate digits in backtracking or initialize literal phases in CDCL SAT solvers. Crucially, the neural component changes search order without pruning feasible assignments, so the correctness and completeness guarantees of the underlying solver are preserved. On the paper's 9×9 Sudoku set, the SE-RRM itself solves 91.1% of instances; neural guidance reduces median wall-clock time by $33.251\times$ for backtracking and $1.699\times$ for Glucose 4.1, whereas CaDiCaL shows no significant median improvement and a small mean slowdown. The contrast supports the paper's central condition for useful guidance: benefits are strongest when runtime is search-dominated and the solver can recover from imperfect hints, while fixed overhead can erase the gain. Because SE-RRM inference is precomputed and excluded from the reported wall-clock measurements, these results isolate solver-side guidance rather than end-to-end system latency. G-RRM is therefore best viewed as an application-level neuro-symbolic composition: the recurrent model supplies a search prior, while the exact solver retains formal guarantees.

RTM (Esmaeilzadeh et al., 2026) transfers recursive refinement to continuous image generation. It replaces the conventional single-pass MLP mapping network in IMLE- and StyleGAN-family generators with a Recursive Token Mapper that repeatedly applies one shared block to map noise vector z to style vector w. Computation is controlled by H outer refinement steps and L inner cycles, with the original noise-derived tokens re-injected during recurrence. The main implementation uses an MLP-Mixer-style shared block and differentiates only through the final recursive step to keep training memory tractable. In controlled StyleGAN2 comparisons where only the mapper changes, RTM lowers CIFAR-10 FID from 3.88 to 3.55; with StyleGAN2-ADA on AFHQ-v1, it lowers FID from 4.99 to 4.79 while increasing recall from 0.507 to 0.565. The same mapper also improves the matched RS-IMLE pipeline across CIFAR-10, CelebA-HQ, and the

paper's few-shot benchmarks. Increasing H beyond the training depth can yield modest additional FID improvements before plateauing. Although each image is still rendered through the synthesis network once, the mapper itself performs recurrent computation. RTM therefore belongs in the Application Extensions branch because its principal contribution is adapting weight-shared recursive refinement to a continuous generative mapping problem, not introducing a new trajectory-search mechanism for structured reasoning.

Taken together, these three papers show that the recursive latent-computation design pattern is portable across very different interfaces: autoregressive language generation, formally verified symbolic search, and continuous image synthesis. They also demonstrate why application evidence should remain separate from mechanism-level leaderboards. HRM-Text is evaluated through language benchmarks and pretraining efficiency, G-RRM through solver conflicts and wall-clock speedups, and RTM through FID, precision, recall, and generative coverage. Their common evidence is therefore qualitative at the taxonomy level: parameter-shared recursive computation can serve as a reusable computational primitive beyond the structured-grid settings in which modern RRMs were first developed, but success depends on application-specific objectives, interfaces, and deployment costs.

5 Benchmarks and Evaluation Protocols

Evaluation is unusually consequential for recursive reasoning models because reported accuracy depends not only on the learned transition, but also on the problem family, recursion budget, number of trajectories, augmentation, selection rule, and stopping policy. The same nominal percentage can therefore represent very different capabilities. This section first clarifies what the main benchmarks test, then separates candidate generation from candidate selection, and finally summarizes the cross-paper evidence under its reported protocols. The goal is not to impose a new benchmark suite, but to make the semantics of existing results explicit.

5.1 Representative benchmark families and what they test

The modern RRM literature repeatedly uses a small set of highly structured benchmarks. HRM established Sudoku-Extreme and Maze-Hard as difficult small-sample testbeds, and TRM and many follow-up studies reuse the same task families (Wang et al., 2025; Jolicoeur-Martineau, 2025). ARC-AGI instead evaluates few-shot induction of grid transformations (Chollet, 2019). Pencil Puzzle Bench (PPBench) is a newer, broader benchmark for verifiable constraint reasoning across many puzzle varieties (Waugh, 2026); within the RRM literature reviewed here, it should be viewed as an emerging extension rather than an already standardized shared benchmark. These tasks are useful precisely because correctness can be judged at the level of a complete structured solution, but they probe different kinds of reasoning and should not be treated as interchangeable measures of a single latent capability.

Table 1: Representative benchmark families appearing in current recursive-reasoning studies and the evaluation distinctions that matter for interpreting their results.

Benchmark	Primary capability	Output and correctness	Typical RRM use	Main comparability risks
Sudoku-Extreme	Constraint satisfaction and global consistency	Completed 9×9 grid; a solve is correct only when the full grid matches the unique solution	Small-sample training and exact-solve evaluation; often used for depth, stability, and multi-start studies	Symbol relabeling/augmentation, train–test split, evaluated subset, recursion depth, trajectory width, and selector
Maze-Hard / Maze-Unique	Spatial planning and long-horizon path construction	Path on a 30×30 maze; Maze-Hard requires a valid shortest path, while Maze-Unique changes the instance distribution by enforcing unique solvability	Tests recursive refinement over long paths and stability under additional depth	Maze variant, path-verification rule, augmentation, sequential depth, and whether multiple trajectories are explored
ARC-AGI-1 / ARC-AGI-2	Few-shot rule induction and transformation generalization	Exact output grid for held-out test inputs after a few input–output demonstrations; common reporting includes one- or two-attempt protocols	Used to test generalization from small task-specific datasets and to study augmentation, candidate diversity, and test-time compute	pass@1 versus two-attempt/pass@k reporting, heavy augmentation or ensembling, puzzle-identity conditioning, task variants, and selection procedure
PPBench	Heterogeneous verifiable constraint reasoning	Puzzle-specific structured solution with deterministic rule checking; the original golden benchmark contains 300 puzzles across 20 varieties and supports step-level verification	Recent extension used by PTRM on a filtered subset of puzzle families to study stochastic width scaling	Full versus filtered subset, reported puzzle families, direct versus agentic protocol when comparing LLM baselines, rollout depth/width, perturbation scale, and selector

The benchmark details themselves explain several apparent disagreements in the literature. Sudoku-Extreme uses a unique-solution 9×9 constraint problem, so exact-solve accuracy is a stringent global metric: a nearly correct board still counts as a failure. Maze-Hard instead asks for an optimal path in a 30×30 maze whose shortest path exceeds a difficulty threshold; validity and optimality are therefore part of the correctness criterion (Wang et al., 2025). ARC-AGI tasks provide a few demonstration pairs and ask for exact output grids on one or more test inputs, so performance can change substantially with the number of allowed attempts, data transformations, and candidate-selection policy (Jolicoeur-Martineau, 2025; Roye-Azar et al., 2025). PPBench is valuable for a different reason: its puzzle-specific rules permit deterministic step-level verification, creating a natural testbed for process-level diagnostics (Waugh, 2026). The original PPBench golden set contains 300 puzzles spanning 20 varieties. PTRM does not evaluate that full benchmark: it filters the data to six puzzle families and reports golden-set accuracy on five of them, so its PPBench results should be read as a targeted RRM extension rather than a full-benchmark result (Sghaier et al., 2026).

These differences also limit what can be inferred from cross-task transfer. Strong Sudoku performance is evidence for maintaining global discrete constraints, not automatically for few-shot induction; strong maze performance is evidence for constructing an optimal spatial path, not automatically for symbolic rule discovery. ARC-AGI places greater weight on task induction from a small number of demonstrations, whereas PPBench emphasizes heterogeneous but verifiable constraint systems. The benchmark should therefore be treated as part of the claim being evaluated, rather than as a neutral container for a single notion of "reasoning."

5.2 What the reported metrics measure

For structured reasoning, local token or cell accuracy can conceal globally invalid solutions. We therefore distinguish *exact-solve accuracy*, which requires the complete structured output to satisfy the benchmark's correctness criterion, from local predictive accuracy. Exact solve is the more informative measure for Sudoku, maze, and ARC-style tasks whenever it is available.

Multiple-trajectory inference introduces a second distinction. Let $V_i(\hat{y}) \in \{0,1\}$ denote the task-specific correctness check for instance i, and suppose a model produces the candidate set $\mathcal{C}_i = \{\hat{y}_i^{(1)}, \ldots, \hat{y}_i^{(K)}\}$. The *candidate coverage* at width K is

$$C_K = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\left[\max_{1 \le k \le K} V_i\left(\hat{y}_i^{(k)}\right) = 1 \right]. \tag{20}$$

This quantity asks whether a correct solution exists anywhere in the generated set; it is an oracle measure of generation quality. For a *candidate-preserving selector* S that returns an index $s_i = S(\mathcal{C}_i) \in \{1, \ldots, K\}$, the corresponding selected accuracy is

$$A_{S,K} = \frac{1}{N} \sum_{i=1}^{N} V_i\left(\hat{y}_i^{(s_i)}\right). \tag{21}$$

The difference

$$G_{S,K} = C_K - A_{S,K} \ge 0 \tag{22}$$

is a useful *selection gap*: it separates failures to generate a correct candidate from failures to identify one that has already been generated. This is a survey-level diagnostic rather than a new benchmark metric, but it makes the evidence from PTRM, GSE, GRAM, RIM, and multi-start attractor studies easier to interpret. A method can substantially improve C_K while leaving $A_{S,K}$ nearly unchanged if its selector is miscalibrated; conversely, a candidate-preserving selector cannot recover a solution absent from the set. An aggregation rule that synthesizes a new structured output from several candidates is different: it need not satisfy $A_{S,K} \le C_K$ and should be reported separately rather than folded into $G_{S,K}$.

This decomposition also clarifies several commonly reported quantities. Single-trajectory pass@1 primarily reflects top-ranked or single-run accuracy. A pass@K or oracle best-of-K result primarily reflects candidate coverage, whereas Q-head ranking, learned reweighting, external verification, and full-solution mode selection instantiate different candidate-preserving choices of S. Aggregation that constructs a new output from components of several rollouts should instead be identified as a separate decoding procedure. ARC-AGI's two-attempt score should be labeled explicitly as a benchmark-specific multi-attempt protocol rather than silently compared with single-attempt accuracy. Likewise, a fixed-point residual, halting confidence, guide score, or convergence flag is a control signal unless the study demonstrates that it is calibrated to actual solution correctness.

5.3 A minimum reporting contract for RRM evaluation

Because recursion exposes several independent compute axes, parameter count and final accuracy are insufficient for reproducible comparison. At minimum, an RRM result should specify the quantities in Table 2. The list is intentionally compact: it records the variables that most often change the meaning of a reported score without requiring every paper to adopt the same architecture or benchmark.

Table 2: Minimum reporting contract for interpretable and reproducible RRM evaluation.

Report explicitly	**Why it matters**
Task variant, split, and evaluated subset	Distinguishes, for example, Maze-Hard from Maze-Unique and standard ARC evaluation from paper-specific variants or restricted subsets.
Training data and augmentation	Symmetry transforms, puzzle relabeling, synthetic expansion, and task-identity conditioning can change effective data coverage substantially.
Model size and repeated operator	Separates parameter efficiency from gains caused by a different transition architecture or additional learned modules.
Training recursion and gradient horizon	Distinguishes executed recurrent depth from the portion of the trajectory that receives direct gradient credit.
Inference depth or halting rule	Identifies how much sequential computation is spent on each trajectory and whether stopping is fixed, learned, or residual-based.
Trajectory width K and perturbation process	Separates deeper refinement from parallel exploration and specifies how candidate diversity is created.
Selector, aggregator, or verifier	Determines whether the reported score measures candidate-preserving ranking, full-solution voting, synthesized aggregation, or externally verified selection.
Metric and attempt budget	Makes pass@1, pass@K, two-attempt ARC accuracy, exact solve, and local token/cell accuracy distinguishable.
End-to-end compute and variability	FLOPs or forward passes, wall-clock latency, memory, offline teacher/adaptation cost, and seeds/checkpoints expose efficiency and robustness trade-offs.

A compute-matched comparison should therefore hold as many of these variables fixed as possible and vary only the mechanism under study. When that is not possible, reporting them explicitly still prevents a result from being misread as evidence for a stronger recursive transition when the gain may instead come from greater width, heavier augmentation, a different selector, or additional offline computation. This is especially important for ARC, where test-time augmentation and large voting ensembles can materially change reported performance (Roye-Azar et al., 2025), and for stochastic methods, where total accelerator use increases with the number of trajectories even when parallel execution keeps latency manageable (Sghaier et al., 2026; Corbett et al., 2026).

5.4 Cross-paper evidence under heterogeneous protocols

A single leaderboard obscures important differences in model size, recursion budget, trajectory width, augmentation, selection, and task variants. We therefore organize the shared grid-benchmark evidence by the four mechanism-centered families in our taxonomy. The tables report the values used in the reviewed papers, while short notes flag only protocol differences that materially affect comparison. Unless noted otherwise, ARC-AGI values are pass@2 percentages and Sudoku and maze values are pass@1 exact-solve percentages. The Application Extensions branch is intentionally not forced into these tables: HRM-Text, G-RRM, and RTM use language, solver-efficiency, and image- generation metrics, respectively, and are discussed in Section 4.5. AR-TRM is likewise discussed qualitatively in Section 4.1.2 because its character-level autoregressive tasks do not share this benchmark suite.

Architectural foundations. Table 3 compares methods whose main contribution is the recurrent backbone, shared operator, or structural inductive bias. Parameter count is retained because model scale differs substantially across this family.

Table 3: Architectural foundations on shared structured-grid benchmarks. Values are percentages.

Method	Params	ARC-1	ARC-2	Sudoku	Maze
HRM	27M	40.3	5.0	55.0	74.5
TRM	5–19M	44.6	7.8	87.4	85.3
URM	–	53.8*	16.0*	77.6*	–
Mamba-2 TRM	6.28–13.24M	45.88	–	84.2	80.6
SE-RRM	2M	45.3	7.1	98.84	88.8
RIM	5.5–13.6M	47.5	11.3	89.34	87.7
SR^2 Framework	3.4M	44.3†	6.7†	66.63	93.7

Notes. *URM reports pass@1. †SR^2 uses a two-trial pass@2 ARC protocol.

The architectural results show why the tables should not be read as a strict ranking: methods differ by more than an order of magnitude in parameter count, and operator changes can affect candidate coverage differently from top-ranked accuracy. The next family instead changes the behavior of the recursive dynamics themselves.

Dynamical systems. Table 4 groups methods that target convergence, attractor structure, perturbation robustness, or adaptive fixed-point behavior. Here the executed depth and task variant are often as important as the nominal model size.

Table 4: Dynamical-systems methods on shared structured-grid benchmarks. Values are percentages.

Method	Params	ARC-1	ARC-2	Sudoku	Maze
CMM	0.26–5M	–	–	93.7	82.2
EqR	–	–	–	99.8	93.0*
FPRM	7M	47.5	6.2	94.2	87.0
Augmented HRM	27M	–	–	96.9	93.3
TRM + Init. Perturbation	5M	–	–	97.8†	–

Notes. *EqR reports Maze-Unique. †TRM + Init. Perturbation uses 50 perturbed starts on the first 10,000 Sudoku-Extreme tests, with ACT-confidence selection.

These results emphasize a distinct question from architectural scaling: whether additional recursive steps remain stable and useful. In contrast, trajectory-search methods deliberately spend extra compute on alternative solutions rather than only extending one path.

Trajectory search. Table 5 contains methods that allocate test-time compute to multiple trajectories, particles, or task-specific adaptation. Their scores therefore depend on the trajectory width or adaptation budget as well as the underlying recursive operator.

Table 5: Trajectory-search methods on shared structured-grid benchmarks. Values are percentages.

Method	Params	ARC-1	ARC-2	Sudoku	Maze
GRAM	10M	52.0	11.1	97.0	–
DRM	7–14M	50.5*	9.6†	–	–
PTRM	5–7M	–	9.72	98.75	86.73
GSE	7M	–	–	98.0	85.3
TTA-TRM	7M	–	6.67	–	–

Notes. *DRM uses ARC-Easy / ARC-AGI-1-style evaluation. †Its ARC-2 value is from ARC2-Eval.

Because width and adaptation can increase test-time work in different ways, these numbers should not be used to claim a better single-trajectory transition. The fourth mechanism-centered family shifts the intervention back to training and supervision.

Adaptive supervision. Table 6 groups methods that primarily change the training signal, curriculum, or target construction. Their evaluation should therefore be interpreted together with any extra teacher or optimization cost.

Table 6: Adaptive-supervision methods on shared structured-grid benchmarks. Values are percentages.

Method	Params	ARC-1	ARC-2	Sudoku	Maze
CGAR	5M	–	–	86.02	–
DIS	0.8–7M	41.3	6.0	–	–
RSM	2.5–5M	–	–	97.5	∼80.0
TRM-Planner	3.55–7M	48.13	9.17	82.77	–

The adaptive-supervision results illustrate that lower online recursion or a simpler deployment path may still rely on additional training-time or offline computation.

Application evidence. No shared performance table is constructed for the Application Extensions branch. HRM-Text reports language-model accuracy together with pretraining tokens and FLOPs; G-RRM reports symbolic-solver conflicts and wall-clock speedups; and RTM reports FID, precision, recall, and related generative metrics. Combining these quantities into a single table would recreate the same false-leaderboard problem that motivated the taxonomy-based split. Their quantitative evidence is therefore synthesized in Section 4.5, while the tables here remain restricted to protocols with at least partially shared structured benchmarks.

Efficiency claims are reported separately but in the same subsection so that the accuracy–compute trade-off remains visible.

Efficiency trade-offs. Table 7 summarizes the two efficiency studies that report compact accuracy–resource trade-offs: LASER targets training activation memory, whereas Shortcut Models target inference-step reduction. Quantization is not inserted into the same table because bit width and scaling granularity are not commensurate with memory reduction or recursive-step FLOPs; its recursion-specific failure mode is synthesized in Section 4.4.2. The tabulated values should therefore be read within their respective experimental settings rather than as a direct ranking.

Table 7: Efficiency comparison of recursive models.

Method	Task	Efficiency Focus	Acc. (%)	Baseline (%)
LASER*	Maze	Training activation memory	98.5	97.8
Shortcut Models†	ARC-AGI-1	Inference step reduction	28.6–39.7	40.4

* Memory: 2.85 GB → 1.14 GB (∼60% lower). † 8-/4-/2-hop variants: 0.125 × /0.25 × /0.5× baseline FLOPs.

Taken together, the evidence supports comparison along separate axes rather than a single score: operator quality, stable depth, trajectory width, selection, supervision, and compute. The reporting contract above makes this principle operational: a benchmark number is interpretable only together with the inference and selection budget that produced it.

6 Comparison and Discussion

The reviewed papers show that recursive reasoning is not a single architecture, but a shared design pattern: a learned operator is repeatedly applied to an internal state so that additional computation can refine, repair, diversify, or select a solution. This makes recursion attractive for structured reasoning because it decouples parameter count from effective computational depth. At the same time, repeated application can amplify instability, miscalibration, poor selection, or train–test mismatch. Section 5 showed why these effects must be interpreted together with the benchmark and inference protocol that expose them.

A useful mechanism-level comparison should therefore focus less on which method has the highest reported score and more on which part of the recursive reasoning pipeline each paper changes: the repeated operator, the training contract, the allocation of depth or width, the stability objective, or the deployment-time controller. Application papers require one additional distinction: whether the recurrent core itself changes or whether the novelty lies primarily in the surrounding task interface, objective, decoder, or symbolic system.

6.1 Recursive mechanisms: operator, depth, and width

HRM and TRM establish the core latent-recursion paradigm with Transformer-style operators. Later work modifies this operator in different directions: URM strengthens recurrent nonlinear processing, Mamba-2 TRM replaces the block with a state-space/attention hybrid, SE-RRM encodes symbol equivariance, and RIM reframes recursive updates as inference-machine operations with proposal and reweighting components.

A second distinction is between depth and width. Depth scaling runs one trajectory for more recursive steps, as in TRM, RSM, CMM, and EqR. Width scaling samples or constructs multiple trajectories and selects among them, as in PTRM, GSE, and GRAM (see Figure 5). Depth helps when a trajectory can keep improving toward a solution-bearing attractor; width helps when deterministic recursion gets trapped or when multiple plausible trajectories should be explored. Mamba-2 TRM makes this distinction clear: stronger pass@k gains than pass@1 gains suggest better candidate coverage rather than better top-ranked selection.

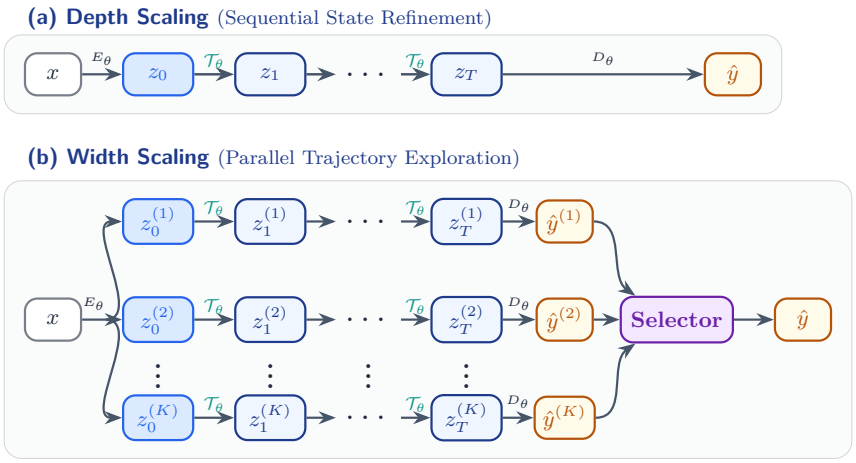

Figure 5: Conceptual comparison of test-time compute scaling mechanisms. Depth scaling (a) applies the parameter-shared transition operator $\mathcal{T}_\theta$ sequentially to refine a single latent trajectory starting from input x. Width scaling (b) unrolls K parallel trajectories using the same shared transition operator, decodes each terminal state with D_θ, and then combines the resulting candidate predictions through a selector or aggregation rule.

6.2 Training, credit assignment, and stability

The training contract determines what the repeated operator learns to be: a local corrector, a denoiser, a fixed-point iterator, a policy-improvement step, a stochastic proposal, or an attractor map. HRM and TRM rely on deep supervision and detached state carry. CGAR schedules recursion depth during training. RSM uses detached warm-up computation with final-step loss. DRM and SPRM introduce denoising or perturbation-based curricula. DIS adds step-wise improvement targets through a policy-improvement interpretation.

Stability is therefore central. Extra recursion is useful only if the learned dynamics remain solution-aligned. RSM treats settling behavior as a reliability signal, CMM explicitly regularizes the recursive dynamics toward stable equilibria, EqR links scalable inference to attractor convergence, and GSE diagnoses when stochastic exploration is likely to help through stability and guide-alignment signals. Future work should report not only accuracy but also convergence traces, fixed-point residuals, halting calibration, and failure cases.

6.3 Reliability, selection, and deployment

The main open issue is reliability. More depth can improve a solution, but it can also move the model toward a wrong fixed point. More sampled trajectories can improve coverage, but they help only if the correct candidate can be selected. This makes selection a core part of recursive reasoning rather than a secondary implementation detail.

PTRM uses the existing Q-head for selection, GSE studies guide alignment explicitly, RIM introduces a learned reweighting mechanism, and GRAM samples multiple latent trajectories but still requires scoring, aggregation, or verification. This distinction is especially important for ARC-style evaluation, where pass@1, pass@2, and pass@k measure different capabilities. A model may generate the correct solution somewhere in its candidate set while still failing to rank it first.

The word "reasoning" should also be used carefully. ARC-AGI stresses few-shot rule induction, Sudoku stresses constraint satisfaction, maze solving stresses spatial planning, and character-level algorithmic tasks stress sequential formal generalization. A method that improves one setting does not automatically solve the others. The Application Extensions branch makes this dependence explicit: HRM-Text pays for recurrent depth inside an autoregressive serving stack, G-RRM must account for neural inference in addition to symbolic search, and RTM trades extra mapper recursion against generative quality and coverage. Deployment-ready recursive reasoners will therefore need not only a strong recursive prior, fast adaptation, calibrated selection, and an inference controller that knows when to stop, sample, verify, or abstain, but also application-specific accounting of latency, memory, and external-system cost.

6.4 Evidence synthesis: what current results do and do not establish

The reviewed literature supports several cross-paper conclusions, but their evidential status is not uniform. Table 8 distinguishes findings that are supported under current protocols from claims that remain unestablished or are contradicted by existing diagnostics. The assessment is intentionally qualitative rather than a statistical meta-analysis: datasets, training recipes, recursion budgets, trajectory counts, and selection rules differ too substantially to justify pooled effect estimates.

At a high level, three conclusions are more stable than the individual leaderboard numbers. First, parameter-shared recursion is a viable way to allocate internal compute, but the existing evidence does not make hierarchy itself the essential ingredient. Second, additional test-time compute is useful only when the recursive dynamics or stochastic exploration remain solution-aligned; neither large depth nor stable convergence is sufficient by itself. Third, width-based scaling separates the ability to *generate* a correct candidate from the ability to *select* it, so both quantities should be measured. The least settled issue is transfer: strong results remain concentrated on structured benchmarks, making broad claims about general reasoning premature.

Table 8: Survey-level evidence synthesis across the reviewed RRM literature. The assessment describes what the cited studies support under their reported protocols; it is not a universal claim beyond those settings.

Claim	Assessment	Evidence and limitation
Parameter-shared latent recursion is a viable compact reasoning primitive.	Supported on current structured benchmarks.	HRM and TRM obtain strong results with small recurrent backbones, and later operator variants preserve the shared-recursion scaffold (Wang et al., 2025; Jolicoeur-Martineau, 2025; Gao et al., 2025; Wang & Reid, 2026). However, supervision, augmentation, task embeddings, and inference budget often co-vary, and most evidence remains puzzle-centric.
A hierarchical two-timescale backbone is necessary for effective recursive reasoning.	Not established.	TRM removes most explicit hierarchy while remaining competitive, and mechanistic studies show that functional roles can arise from update schedules and input asymmetry (Jolicoeur-Martineau, 2025; Miyanishi & Morimura, 2026; Shen et al., 2026). Broad compute- and training-matched ablations isolating hierarchy are still missing.
Increasing recursive depth is monotonically beneficial.	Not supported as a general rule.	Mechanistic analyses report abrupt transitions, diminishing returns, and incorrect attractors; EqR shows that very large depth can help when attractors are solution-aligned, while FPRM uses convergence-aware halting to adapt compute (Ren & Liu, 2026; Huang et al., 2026; Movahedi et al., 2026). The conclusion is conditional rather than negative: some tasks benefit greatly from more depth.
Increasing trajectory width can recover solutions missed by a single trajectory.	Supported, but domain-limited.	PTRM and GSE improve structured-task accuracy through stochastic exploration, while attractor-based interventions show that changed initial conditions or model realizations can escape unproductive basins (Sghaier et al., 2026; Corbett et al., 2026; Ren & Liu, 2026; Efstathiou & Balwani, 2026). Compute grows with width, and gains depend on perturbation design and selection.
Convergence or a small fixed-point residual certifies answer correctness.	Contradicted as a general claim.	HRM analyses identify multiple fixed points, including incorrect ones; EqR explicitly reports spurious low-residual attractors; and FPRM uses fixed-point residuals as an adaptive halting signal (Ren & Liu, 2026; Huang et al., 2026; Movahedi et al., 2026). Residuals remain useful for halting and diagnosis, but not as correctness certificates.
Candidate generation and candidate selection are distinct bottlenecks.	Supported by multiple studies.	Mamba-2 TRM improves higher-pass@k coverage more than pass@1; PTRM depends on Q-head selection; GSE exposes guide alignment; and RIM adds learned reweighting (Wang & Reid, 2026; Sghaier et al., 2026; Corbett et al., 2026; Komisarczyk et al., 2026). Coverage and selected accuracy are still reported under heterogeneous attempt budgets and selectors.
Encoding known problem structure can improve robustness or extrapolation.	Supported in task-specific settings.	SE-RRM encodes symbol-permutation equivariance and reports less reliance on augmentation, stronger Sudoku size extrapolation, and competitive ARC performance (Freinschlag et al., 2026). Such gains require identifying an appropriate structural prior in advance.
Current evidence establishes broad transfer from structured puzzles to open-ended or autoregressive reasoning.	Not yet established.	HRM-Text shows promising language-pretraining transfer, whereas controlled Autoregressive TRM experiments find no reliable gain from the full TRM-style architecture on their character-level tasks (Wang et al., 2026; Rauba et al., 2026). Objectives and interfaces differ, and some application extensions fall outside the strict RRM definition used here.

7 Open Problems and Future Directions

Depth generalization beyond the training horizon. A motivation behind recursive reasoning is the ability to run a model trained at modest depth for many more steps at inference. This often fails when the latent dynamics drift, oscillate, or settle into spurious fixed points. CMM and EqR make progress, but there is still no general theory explaining when depth extrapolation should work. Future work should connect empirical unroll stability to spectral properties of the transition, contraction behavior, and task-conditioned attractor geometry.

Selection without external verifiers. Many structured tasks have natural verifiers, but general reasoning tasks often do not. While recent attractor-based frameworks like EqR achieve implicit selection by converging to a single stable state, this paradigm is fundamentally restricted to closed-ended tasks with

unique solutions. In open-ended or generative reasoning tasks, traditional internal Q-heads, reweighters, and label-free diagnostics remain underexplored in these multi-modal landscapes. A key open problem is to develop flexible training objectives and selection mechanisms that support multiple valid trajectories, enabling implicit selection to scale effectively in generative or ambiguous reasoning domains without relying on external verifiers.

Noise schedules and trajectory diversity. PTRM, GSE, GRAM, and SPRM show that stochasticity can improve recursive reasoning, but the amount and timing of noise remain open design choices. Static Gaussian perturbations are unlikely to be optimal across tasks and depths. Future models should learn adaptive noise schedules conditioned on state uncertainty, convergence speed, guide alignment, or disagreement among particles. The goal is to balance early exploration with late convergence.

Credit assignment over long recursions. Full backpropagation through many recursive steps is expensive; truncated or detached alternatives are cheaper but may miss long-horizon dependencies. Deep supervision helps but can make intermediate states too greedy. DIS, DRM, RSM, and CGAR all propose partial solutions. A deeper question is how to train a transition operator whose intermediate states are useful without forcing every intermediate state to resemble the final answer too early.

Operator design and normalization. The field has not settled on the best repeated operator. Attention, Mamba-style sequence models, convolutional nonlinear blocks, equivariant layers, and inference-inspired modules each have strengths. Normalization is especially important for weight-tied recurrence: a design that stabilizes a deep feed-forward Transformer may not be optimal when the same block is applied repeatedly. Systematic studies of Pre-LN, Post-LN, residual scaling, gating, and scale normalization under recurrence are needed.

Unifying abstract and algorithmic reasoning capabilities. Current recursive architectures are predominantly specialized, excelling either at abstract rule induction or sequential algorithmic execution, but rarely both. This capability gap suggests that existing recurrent operators rely on domain-specific priors rather than achieving a unified system-2 reasoning mechanism. A critical open challenge is designing recursive models capable of synthesizing both requirements within a single architecture. Future models must possess the expressive flexibility to first induce implicit abstract principles from sparse context and seamlessly transition to executing precise, multi-step algorithmic processes within the same latent trajectory.

Interpretability of weight-tied latent trajectories. While recursive models preserve latent expressivity by bypassing explicit language traces, understanding their internal computations remains an open challenge. Current mechanistic interpretability tools, such as sparse autoencoders (SAEs) and activation patching, are designed for feedforward architectures with static, layer-wise representations. In weight-tied recurrence, however, the same latent space is repeatedly reused across cycles, causing individual features to shift in semantic meaning over time (temporal polysemanticity). Future work should develop dynamic interpretability frameworks capable of tracing feature evolution across recursive steps. This requires formalizing causal information flow within cyclic computation graphs, defining metrics for temporal feature stability, and training step-aware dictionary learning models to isolate how specific constraints are resolved at different stages of the recurrence.

Theory of recursive computation. Finally, the field needs a stronger theoretical account of what recursive reasoning models can compute, how their expressivity scales with recursion depth, and how stability constraints affect that expressivity. Existing links to implicit models, dynamical systems, policy improvement, diffusion, and Sequential Monte Carlo are promising but fragmented. A mature theory should explain not only why recursion helps, but also when it fails and how to design transitions that remain both expressive and stable.

Mechanistic validation and inference-time control. The attractor-landscape analysis of TRM is currently limited to the 5M-parameter model on Sudoku-Extreme, leaving open whether the same initialization-dependent failure modes generalize across tasks and recursive architectures. The PCA, sparse-autoencoder,

and loss-trajectory evidence is primarily diagnostic; controlled latent interventions are needed to determine which states or directions causally govern convergence to correct or spurious attractors. The initialization-perturbation strategy should also be evaluated on complete test sets under compute-matched budgets, with explicit ablations on noise scale, number of trajectories, recursive depth, and ACT-confidence calibration. A promising extension is an adaptive controller that detects stalled trajectories and triggers perturbation or restart only when needed.

Fixed-point reasoning beyond symbolic benchmarks. FPRM shows that fixed-point residuals can provide an intrinsic and adaptive halting signal, but its convergence guarantees remain conditional, and some trajectories may oscillate or exhaust the available compute budget. Moreover, a small residual establishes consistency with the learned update operator, not necessarily correctness of the predicted solution. Future work should develop stronger criteria for solution-aligned convergence, improve the computational efficiency of per-example halting, and study whether the same stability and adaptivity extend beyond Transformer-based algorithmic tasks to natural-language and other non-algorithmic domains.

Learning and tracking interaction locality. Current interaction-locality analyses are limited to released checkpoints, manually specified task neighborhoods, and a small embodied-reasoning evaluation. Future work should track SAE features, finite-noise intervention effects, and cross-cycle information propagation throughout training rather than only after convergence. Such analyses could reveal when local correction, segment-level communication, and broader integration emerge, and whether these changes coincide with accuracy improvements, convergence, or stable recursive routines. The framework should also be extended beyond box-level Sudoku and connected-object neighborhoods to richer relational structures, including Sudoku row and column constraints, scene graphs, contact graphs, object trajectories, and temporal or egocentric geometries. A further direction is to make interaction locality an explicit training objective by regularizing each recursive phase toward an intended spatial or relational interaction scale. Accuracy-matched training sweeps would then be needed to determine whether locality-aware objectives improve sample efficiency, robustness, transfer, or depth generalization, and to separate these effects from differences in architecture or training compute.

Time-conditioned scheduling for deep recursion. Adjacent looped-Transformer work already provides a useful clue: LoopFormer (Jeddi et al., 2026) conditions recurrent updates on normalized time and step size and trains across variable-length trajectories so that the compute budget becomes an explicit input to the recurrent dynamics. The open question for HRM/TRM-style structured RRMs is therefore sharper than simply adding a step index. Future work should test whether time-conditioned updates, adaptive noise schedules, or both can prevent shallow effective recursion and premature basin commitment while preserving depth extrapolation and selector calibration. Such comparisons should hold the transition architecture and total inference budget fixed so that improvements can be attributed to scheduling rather than additional compute.

8 Limitations

This survey makes several deliberate choices that constrain the conclusions that can be drawn from it. First, the operational RRM definition is narrower than the full latent-reasoning or loop-model landscape. That restriction is useful for comparing parameter-shared fixed-instance computation, but it necessarily places some recurrent-depth language models, external recursive systems, diffusion-style refinement methods, and other hybrid architectures outside the core class even when they share important mechanisms. The Application Extensions branch reduces this boundary problem but does not eliminate it.

Second, the literature snapshot ends on July 31, 2026, and a large fraction of the modern RRM literature is available primarily as rapidly revised preprints. Publication status, implementation details, and reported numbers may therefore change after the cutoff, and newly released work may alter the balance of evidence. We retain preprints because excluding them would remove much of the current field, but we avoid treating venue status or isolated leaderboard results as evidence of a general mechanism.

Third, we do not rerun the reviewed models under one common training and evaluation stack, nor do we perform a statistical meta-analysis. The survey-level assessments in Section 6.4 therefore synthesize the strongest available controlled comparisons and mechanistic diagnostics rather than estimate pooled effect sizes. Differences in augmentation, data generation, recursion depth, trajectory width, selector use, parameter count, and compute accounting remain important confounders. This is precisely why Section 5 emphasizes protocol-aware reporting, but that reporting contract cannot retroactively normalize studies that omit some of the required information.

Finally, current evidence is concentrated on structured tasks with exact or nearly exact verification, especially Sudoku, mazes, ARC-style grids, and related algorithmic puzzles. These settings are valuable because intermediate and final correctness can often be checked, but they may overstate the generality of mechanisms that benefit from strong structure, augmentation symmetries, or external verification. Claims about open-ended language reasoning, multimodal reasoning, and deployment-scale robustness should therefore be treated as hypotheses for future work rather than established consequences of current RRM results.

9 Conclusion

Recursive reasoning models provide a compact way to allocate additional computation to structured reasoning problems without externalizing every intermediate step into natural-language tokens. This paradigm has diversified into four mechanism-centered families (architectural foundations, dynamical systems, trajectory search, and training optimization) and an application-extension branch that demonstrates transfer to language pretraining, neuro-symbolic solving, and continuous image generation. Across the reviewed evidence, three conclusions are comparatively stable: parameter-shared recursion is a viable compute-allocation primitive even though explicit hierarchy is not established as necessary; additional depth or width is useful only when the resulting dynamics, exploration, and selection remain solution-aligned; and candidate generation must be separated from candidate selection when interpreting test-time scaling. By contrast, monotonic benefits from depth, correctness guarantees from convergence, and broad transfer to open-ended reasoning are not established by the current literature. Progress will therefore depend less on isolated benchmark wins and more on standardized protocols that separate operator quality, depth scaling, width scaling, candidate coverage, selector calibration, application-level compute, and robustness under distribution shift. In particular, reporting candidate-set coverage together with selected accuracy can reveal whether a gain comes from generating better trajectories or from choosing among them more reliably.

References

Arip Asadulaev, Rayan Banerjee, Fakhri Karray, and Martin Takac. Deep improvement supervision. *arXiv preprint arXiv:2511.16886*, 2025.

Euijin Baek, Housam Babiker, Mi-Young Kim, and Randy Goebel. TRM-Planner: Offline target planning and distillation for tiny recursive models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 27058–27070, 2026a.

Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning. *arXiv preprint arXiv:2605.19376*, 2026b.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20232–20242, 2022.

Jay Bear, Adam Prügel-Bennett, and Jonathon Hare. Rethinking deep thinking: Stable learning of algorithms using lipschitz constraints. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Timo Bertram, Sidhant Bhavnani, Richard Freinschlag, Erich Kobler, Andreas Mayr, and Günter Klambauer. G-RRM: Guiding symbolic solvers with recurrent reasoning models. *arXiv preprint arXiv:2607.02491*, 2026.

Ege Çakar, Ketan Ali Raghu, and Lia Zheng. LASER: Low-rank activation SVD for efficient recursion. *arXiv preprint arXiv:2604.17224*, 2026. Accepted to the Latent and Implicit Thinking Workshop at ICLR 2026.

Chris Cameron, Wangzheng Wang, Nikita Ivanov, Ashmita Bhattacharyya, Didier Chételat, and Yingxue Zhang. One step forward and K steps back: Better reasoning with denoising recursion models. *arXiv preprint arXiv:2604.18839*, 2026.

Xinghao Chen, Anhao Zhao, Heming Xia, Xuan Lu, Hanlin Wang, Yanjun Chen, Wei Zhang, Jian Wang, Wenjie Li, and Xiaoyu Shen. Reasoning beyond language: A comprehensive survey on latent chain-of-thought reasoning. *arXiv preprint arXiv:2505.16782*, 2025.

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

Andrew Corbett, Archit Sood, Anna Tzatzopoulou, Sai-Aakash Ramesh, and Tim Dodwell. Boosting inference with guided reasoning: Stochastic exploration for recursive models. *arXiv preprint arXiv:2605.25230*, 2026.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Yunlong Deng, Boyang Sun, Yan Li, Lingjing Kong, Zeyu Tang, Kun Zhang, and Guangyi Chen. Selection, reflection and self-refinement: Revisit reasoning tasks via a causal lens. *arXiv preprint arXiv:2510.08222*, 2025.

Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

Andreas Efstathiou and Aishwarya Balwani. Recursive reasoning as attractor landscape search: Mechanistic dynamics of the tiny recursive model. In *International Conference on Learning Representations*, 2026.

Vasiliy A. Es'kin and Mikhail E. Smorkalov. Dynamical systems theory behind a hierarchical reasoning model. *arXiv preprint arXiv:2603.22871*, 2026.

Mehdi Esmaeilzadeh, Alexia Jolicoeur-Martineau, Chirag Vashist, and Ke Li. One pass is not enough: Recursive latent refinement for generative models. *arXiv preprint arXiv:2605.15309*, 2026.

Richard Freinschlag, Timo Bertram, Erich Kobler, Andreas Mayr, and Günter Klambauer. Symbol-equivariant recurrent reasoning models. *arXiv preprint arXiv:2603.02193*, 2026.

Zitian Gao, Lynx Chen, Yihao Xiao, He Xing, Ran Tao, Haoming Luo, Joey Zhou, and Bryan Dai. Universal reasoning model. *arXiv preprint arXiv:2512.14693*, 2025.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Navid Hakimi. Form follows function: Recursive stem model. *arXiv preprint arXiv:2603.15641*, 2026.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

Benhao Huang, Zhengyang Geng, and Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning. *arXiv preprint arXiv:2605.21488*, 2026.

Thorir Mar Ingolfsson, Wajeeha Tahir, Anna Tegon, Lionnus Kesting, Gamze İslamoğlu, and Luca Benini. Quantizing recursive reasoning models. *arXiv preprint arXiv:2607.16237*, 2026.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations*, 2026.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Mieszko Komisarczyk, Saurabh Mathur, Maurice Kraus, Sriraam Natarajan, and Kristian Kersting. Recursive inference machines for neural reasoning. *arXiv preprint arXiv:2603.05234*, 2026.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamile Lukosiute, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Graves Larson, Sam McCandlish, Sandipan Kundu, Saurav Kadavath, Shannon Yang, Thomas Henighan, Timothy Maxwell, Timothy Telleen-Lawton, Tristan Hume, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Measuring faithfulness in chain-of-thought reasoning. *arXiv preprint arXiv:2307.13702*, 2023.

Ronan Killian McGovern. Test-time adaptation of tiny recursive models. *arXiv preprint arXiv:2511.02886*, 2025.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

Yosuke Miyanishi and Tetsuro Morimura. Interaction locality in hierarchical recursive reasoning. *arXiv preprint arXiv:2605.20784*, 2026.

Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T. Konstantin Rusch, and Antonio Orvieto. Fixed-point reasoners: Stable and adaptive deep looped transformers. *arXiv preprint arXiv:2606.18206*, 2026.

Kaleem Ullah Qasim and Jiashu Zhang. Accelerating training speed of tiny recursive models via curriculum guided adaptive recursion. *arXiv preprint arXiv:2511.08653*, 2025.

Paulius Rauba, Claudio Fanconi, and Mihaela van der Schaar. Tiny autoregressive recursive models. *arXiv preprint arXiv:2603.08082*, 2026.

Zirui Ren and Ziming Liu. Are your reasoning models reasoning or guessing? a mechanistic analysis of hierarchical reasoning models. *arXiv preprint arXiv:2601.10679*, 2026.

Antonio Roye-Azar, Santiago Vargas-Naranjo, Dhruv Ghai, Nithin Balamurugan, and Rayan Amir. Tiny recursive models on ARC-AGI-1: Inductive biases, identity conditioning, and test-time compute. *arXiv preprint arXiv:2512.11847*, 2025.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 6695–6706, 2021.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model. *arXiv preprint arXiv:2605.19943*, 2026.

Shiv Shankar. Learning shortcut models for efficient recursive reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pp. 1351–1360, 2026.

Jucheng Shen, Barbara Su, and Anastasios Kyrillidis. One model, two roles: Emergent specialization in a shared recurrent transformer. *arXiv preprint arXiv:2605.17811*, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi-Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Guan Wang, Changling Liu, Chenyu Wang, Cai Zhou, Yuhao Sun, Yifei Wu, Shuai Zhen, Luca Scimeca, and Yasin Abbasi-Yadkori. HRM-Text: Efficient pretraining beyond scaling. *arXiv preprint arXiv:2605.20613*, 2026.

Wenlong Wang and Fergal Reid. Tiny recursive reasoning with Mamba-2 attention hybrid. *arXiv preprint arXiv:2602.12078*, 2026.

Justin Waugh. Pencil puzzle bench: A benchmark for multi-step verifiable reasoning. *arXiv preprint arXiv:2603.02119*, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, Irwin King, Xue Liu, and Chen Ma. A survey on test-time scaling in large language models: What, how, where, and how well? *arXiv preprint arXiv:2503.24235*, 2025.

Rui-Jie Zhu, Tianhao Peng, Tianhao Cheng, Xingwei Qu, Jinfa Huang, Dawei Zhu, Hao Wang, Kaiwen Xue, Xuanliang Zhang, Yong Shan, Tianle Cai, Taylor Kergan, Assel Kembay, Andrew Smith, Chenghua Lin, Binh Nguyen, Yuqi Pan, Yuhong Chou, Zefan Cai, Zhenhe Wu, Yongchi Zhao, Tianyu Liu, Jian Yang, Wangchunshu Zhou, Chujie Zheng, Chongxuan Li, Yuyin Zhou, Zhoujun Li, Zhaoxiang Zhang, Jiaheng Liu, Ge Zhang, Wenhao Huang, and Jason Eshraghian. A survey on latent reasoning. *arXiv preprint arXiv:2507.06203*, 2025.

Appendix

A Notation and Mathematical Conventions

The appendix collects the unified mathematical notation used in the survey. Table 9 summarizes the common notation used across our formalization and representative model examples.

Table 9: Unified notation used throughout the survey.

Symbol	Term / Concept	Definition / Context
Spaces and core variables		
$\mathcal{X}, \mathcal{Y}, \mathcal{Z}$	Input, output, state spaces	Spaces for instances, outputs, and recursive states.
$x,\ y,\ \hat{y}$	Instance, target, prediction	Input instance, target output, and final prediction.
$\theta,\ \mathcal{M}_\theta$	Parameters and RRM	Learned parameters and the complete recursive model.
e_x	Encoded context	Input-conditioned context produced by E_θ.
$z_t,\ z_{0:T},\ z_T$	State and trajectory	State at step t, full trajectory, and terminal state.
Operators and neural dynamics		
$E_\theta,\ \mu_\theta$	Encoder and initializer	Encode x and initialize z_0.
$\mathcal{T}_\theta$	Shared transition kernel	Reused operator mapping the current state to the next state.
$D_\theta,\ p_\theta$	Decoder and predictive distributions	D_θ defines the output distribution conditioned on the terminal state; p_θ denotes predictive or other method-specific output distributions.
$\mathbb{P}_\theta(\cdot \mid x)$	Trajectory law	Probability measure over recursive trajectories induced by μ_θ and $\mathcal{T}_\theta$.
$F_\theta,\ \delta_a$	Deterministic update, Dirac measure	Deterministic transition and its point-mass representation.
$H_\theta,\ \tau$	Halting rule, stopping time	Optional adaptive stopping controller and stopping step.
$\arg\max$	Decision operator	Selects the highest-probability output.
Compute, depth, and width		
$t,\ T,\ T_{\max}$	Step and depth budgets	Current step, executed depth, and maximum depth.
$K,\ z_{0:T}^{(k)}$	Width and trajectory sample	Number of trajectories and the kth trajectory.
pass@k	Candidate-coverage metric	Success within k attempts.
Representative architecture-specific notation		
$\tilde{x},\ z_L,\ z_H$	HRM input and states	Encoded input and HRM low-/high-level states.
f_I, f_L, f_H, f_O	HRM modules	HRM encoder, state updates, and output decoder.
$N,\ c,\ j$	HRM cycle indices	Number of cycles and their high-/low-level indices.
$f_\theta,\ y_t,\ z_t^{(r)},\ n$	TRM refinement notation	Shared core, solution state, inner state, and inner-step count.