

A Survey of Improving Reasoning in LLM

Mechanisms, Learning, Control, Evidence, and Vision

SHIROU JING, Rice University, USA

CHUAN LIU, Rice University, USA

RUIBING SONG, Rice University, USA

YU HU, Rice University, USA

YIXIN HE, Rice University, USA

TONY GENG, Rice University, USA

Recent advances in reasoning have transformed large language models into systems capable of complex problem solving. However, progress in reasoning systems has increasingly outpaced the development of a systematic understanding of how reasoning should be designed. We address this gap by treating extra training complexity beyond standard autoregressive training as a unifying abstraction. We operationalise this abstraction through three coupled design questions: what reasoning computation is made available, how it is learned, and how its use is controlled. Together, these questions define the mechanism, learning, and control layers, with mechanism and learning forming a 3×4 landscape and control cutting across their combinations. By placing otherwise separate methods within a common structure, this framework allows their findings to accumulate into systematic knowledge about how reasoning systems should be designed. We further assess the landscape through a five-level evidence hierarchy, moving from whether a complete recipe works to whether gains are design-attributed, newly reachable, and transferable. Together, the structural map and evidence analysis connect design patterns to evidential strength and support cumulative principles of reasoning design. From this synthesis, we advance a vision in which co-design across mechanism, learning, and control becomes a principle for the next generation of reasoning systems.

CCS Concepts: • **Computing methodologies** → **Natural language processing**; *Reinforcement learning*; *Neural networks*.

Additional Key Words and Phrases: large language models, reasoning, extra training complexity, recurrent processing, intermediate-state insertion, objective reshaping, learning signals, computation control, chain-of-thought, latent reasoning

ACM Reference Format:

Shirou Jing, Chuan Liu, Ruibing Song, Yu Hu, Yixin He, and Tony Geng. 2026. A Survey of Improving Reasoning in LLM: Mechanisms, Learning, Control, Evidence, and Vision. *ACM Comput. Surv.* 1, 1 (October 2026), 34 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 INTRODUCTION

Large language models have rapidly evolved from general-purpose text predictors into systems capable of solving increasingly demanding reasoning tasks. Recent reasoning-oriented models have shown substantial gains in mathematics, coding, scientific problem solving, and multi-step decision making [30, 32, 134, 205], accompanied by a growing family of models that explicitly devote more effort to difficult problems before producing an answer. These advances

Authors' addresses: Shirou Jing, Rice University, Houston, TX, USA; Chuan Liu, Rice University, Houston, TX, USA; Ruibing Song, Rice University, Houston, TX, USA; Yu Hu, Rice University, Houston, TX, USA; Yixin He, Rice University, Houston, TX, USA; Tony Geng, Rice University, Houston, TX, USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

have made reasoning a practical capability, not only a benchmark objective. Stronger reasoning now supports applications in which language models plan over extended horizons, use tools, write and debug software, interact with external environments, and coordinate multi-step workflows [146, 155, 189, 206]. As these systems are asked to operate with greater autonomy and solve problems that cannot be answered through immediate pattern completion, improving their reasoning capability has become a central objective in the development of modern language models.

The field has pursued this objective through an increasingly diverse set of modelling and training strategies [11, 32, 34, 38, 55, 58, 60, 61, 70, 101, 130, 142, 150, 165, 196, 213, 220, 226, 245]. However, the resulting advances in reasoning systems have increasingly outpaced the development of a systematic understanding of how reasoning should be designed. One reason is that much of this progress has unfolded in an ad hoc manner, with different lines of work developing around their own design choices and assumptions. Existing surveys have provided valuable syntheses of major strands of this literature, including test-time scaling [6, 74, 82, 209, 234], long and latent reasoning [20, 23, 244], reinforcement learning across the model lifecycle [90, 112, 203], and efficient reasoning [147]. What remains missing is a common design abstraction that relates these strands within a shared account of reasoning design. Without such an abstraction, rapid methodological progress does not readily become cumulative design knowledge, leaving advances in reasoning capability disconnected from an equally systematic understanding of how future systems should be built. A common basis for such an abstraction emerges from a recurring shift across these approaches, as many move beyond standard autoregressive training by introducing extra training complexity to deliberately shape how the model reasons. We therefore use extra training complexity as a unifying lens through which the apparent diversity of these methods resolves into three coupled design decisions.

The first decision concerns what training changes in the model’s reasoning process. The autoregressive baseline fixes three aspects of computation that a reasoning mechanism can reopen: depth, sequence positions, and the predictive objective. Reopening them yields recurrent processing, intermediate-state insertion, and objective reshaping, respectively. The second decision concerns how the resulting reasoning process is learned from available supervision. Training stage alone does not answer this question: pretraining, mid-training, and post-training primarily determine which supervision is available, not how that supervision shapes reasoning [159, 227]. We therefore distinguish learning according to the source of supervision: text-derived predictive supervision, self-generated and bootstrapped supervision, reasoning-trace supervision, and outcome and process feedback. Mechanism and learning specify what reasoning computation is available and how it is acquired, but neither determines how that computation should be controlled. This remaining decision gives rise to static, dynamic, and curriculum control. These three decisions are the mechanism, learning, and control layers, shown together in the Coupled Design Layers panel of Figure 1. They are separable enough to define a common design space, but coupled enough that none can be understood in isolation, and that tension is what makes their combinations worth studying systematically.

Among the three coupled layers, mechanism and learning define the core combinatorial design space: three mechanism families crossed with four learning-signal families yield a 3×4 landscape, with control cutting across these combinations to govern how they are used. Mapping the literature into this structure makes reasoning design a systematic object of study, providing a common basis for analysing the field and deriving broader design principles. Turning this structural account into reliable design knowledge also requires asking how far current experiments justify the claims made for these designs. To make that assessment possible, we assemble the model trajectories, datasets, supervision artefacts, and benchmarks needed to determine what reported improvements actually demonstrate. We then organise the evidence into five increasingly strong claims: benchmark gain, matched-resource gain, design-attributed gain,

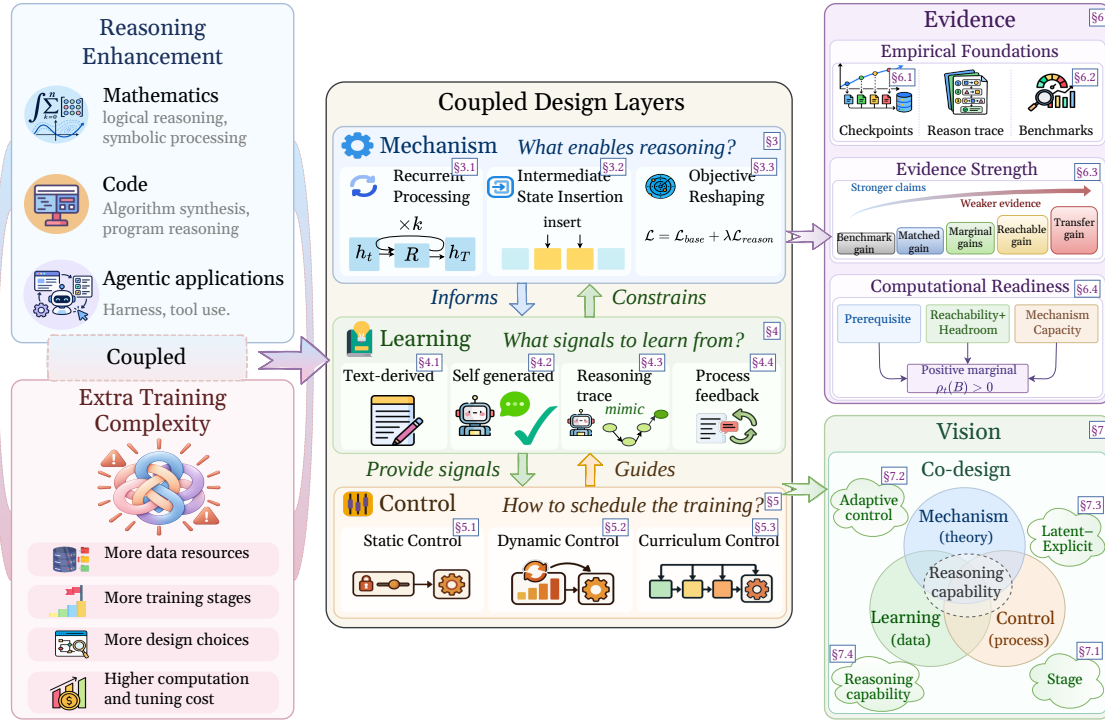


Fig. 1. Overview of extra training complexity as a lens on reasoning enhancement.

newly reachable reasoning [105, 118, 126], and transferable reasoning capability. Together, the design landscape and the evidence framework provide a foundation for cumulative knowledge about how reasoning systems should be built.

By unifying the structure of reasoning design with the evidence that supports it, our framework makes reasoning capability itself a systematic design problem from which field-level principles can be derived. From this synthesis emerges a broader vision of reasoning systems as evolving designs, shown in the Vision column of Figure 1, in which learning across training stages, the interface between latent and explicit computation, and adaptive control are coordinated around the reasoning capabilities being developed. This vision defines a research agenda for reasoning in which co-design becomes the organising principle for building future capabilities.

This survey makes three contributions.

- We introduce *extra training complexity* as a unifying abstraction for reasoning enhancement and decompose it into three coupled design layers: mechanism, learning, and control. This framework turns reasoning enhancement into a systematic design space for cumulative analysis and future qualitative and quantitative study.
- We map existing methods onto a 3×4 mechanism–learning landscape, with control cutting across it, and evaluate this space through a five-level hierarchy of empirical claims. This turns a map of methodological activity into an evidence-grounded foundation for cumulative principles that can guide the design of future reasoning systems.
- We synthesise these structural and empirical insights into a forward-looking vision of reasoning systems as evolving co-designs across mechanism, learning, and control. This vision makes co-design an organising principle for future reasoning research.

Table 1. Comparison with existing surveys. A tick indicates that the dimension is explicitly organised or analysed in a dedicated section or table; incidental discussion does not count.

Survey	Spans training stages	Covers reasoning mechanisms	Covers learning signals	Covers computation control	Provides empirical resources	Relates design dimensions	Structures evidence strength
Frontiers in reasoning [82]	×	×	✓ §4	✓ §5.4.1	×	✓ §5.4.2	×
Test-time scaling [234]	×	✓ §2	✓ §3.1	✓ §5.3	×	×	×
Long chain-of-thought [20]	×	✓ §4.1	✓ §4.2	×	✓ §7	×	×
RL across the lifecycle [112]	✓ §3–4	×	✓ §4.2	✓ §4.4	✓ §5–6	×	×
Efficient reasoning [147]	✓ §4–6	✓ §6.1	✓ §4–5	✓ §3.1	×	×	×
Latent reasoning [244]	×	✓ §3	×	×	×	×	×
This survey	✓ §2.3	✓ §3	✓ §4	✓ §5	✓ §6	✓ Tab. 7	✓ §6.3

1.1 Positioning Against Existing Surveys

Each of those organising axes fixes what a survey can record, and so what its entries can be compared against. One review spans pretraining, mid-training and post-training and organises within them by the optimisation signal [112], so a looped model and a written trace share a chapter whenever reward trains both. Another sorts by the stage at which cost is cut [147], which leaves out work whose purpose is to add computation, and the reviews built on a mechanism follow that mechanism alone, not the learning signal or the control that accompany it. Table 1 sets the six closest side by side on the dimensions each one organises, and on whether it relates those dimensions or grades the evidence behind them. No single dimension belongs to this survey alone. What is new is the relation between them and the grading of the evidence behind them; the design abstraction here is extra training complexity itself, together with the coupled decisions that make a mechanism available, teach it, control its use, and evidence it.

1.2 Scope and Roadmap

We focus on work in which extra training complexity changes how reasoning computation is made available, learned, or controlled. It covers recurrent processing, intermediate-state insertion, objective reshaping, alternative learning signals, and explicit control. We trace it through the closest surveys and their cited and citing work, plus recent papers meeting the same criteria, to September 2026. Counts and citation relations refer to this corpus, not to all reasoning work.

§2 establishes the autoregressive baseline and the supervision available at each training phase. §3 develops the mechanism layer, §4 the learning layer, and §5 the control layer. §6 assembles the empirical foundations and asks what current evidence establishes. The final chapter develops directions for designing the three layers jointly.

2 PRELIMINARIES

This section establishes the computational and supervision preliminaries needed for the mechanism and learning layers that follow. Standard autoregressive language modelling provides the computational baseline (§2.1), relative to which we distinguish three recurring primitives (§2.2): recurrent processing, intermediate-state insertion, and objective reshaping. We then characterise how supervision availability changes across pretraining, mid-training, and post-training (§2.3), constraining what information can be used to learn these computational changes. These concepts provide the foundation for the mechanism and learning layers developed in §3 and §4.

2.1 Autoregressive Language Modelling as the Baseline

The models this survey covers are autoregressive: they factorise the probability of a sequence left to right. Such a model, with parameters θ , is trained on a corpus $\mathcal{D}$ of token sequences over a vocabulary $\mathcal{V}$. Write one sequence as $x = (x_1, \dots, x_{|x|})$, and index its positions by i . The model is a stack of L layers mapping the sequence to hidden

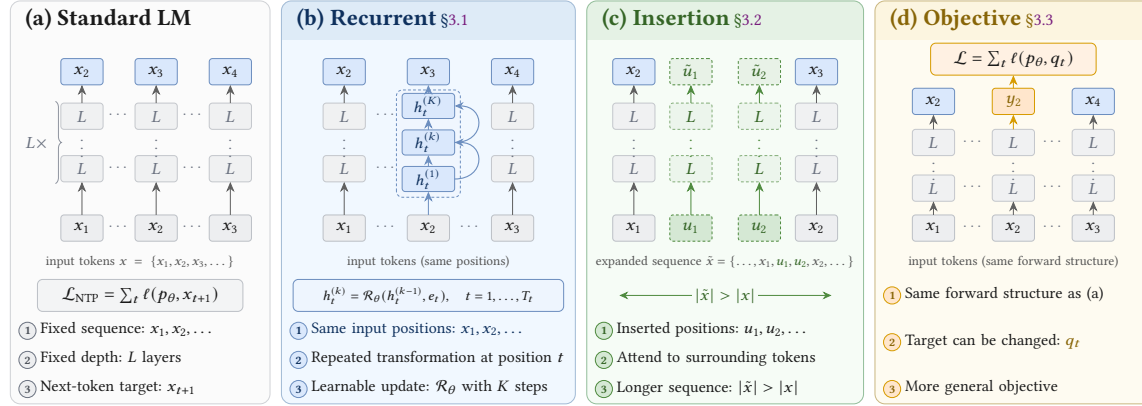


Fig. 2. Three computational primitives against the Standard LM baseline (a): Recurrent (b) reopens its fixed depth, Insertion (c) its fixed sequence, Objective (d) its next-token target.

states $h_i^{(\ell)} \in \mathbb{R}^d$, one per position i and layer ℓ , and reads the distribution over the next token off the last layer, $p_\theta(\cdot \mid x_{<i}) = \text{softmax}(Wh_{i-1}^{(L)})$. Training minimises the negative log-likelihood of every token given its predecessors,

$$\mathcal{L}(\theta) = - \sum_{x \in \mathcal{D}} \sum_{i=1}^{|x|} \log p_\theta(x_i \mid x_{<i}). \quad (1)$$

This standard formulation fixes three aspects of the computational process that the methods surveyed below reopen. It applies the same fixed stack of L transformations at every position, so computational depth does not vary across predictions. It supervises each position only through the observed next token, without explicitly specifying what intermediate computation should produce that prediction. Computation is indexed by the observed token positions supplied by the sequence; no additional positions are introduced solely to support intermediate computation.

The methods in this survey depart from this baseline through training-induced changes to one or more of these properties. These changes are distinct from simply widening the model, which increases capacity while preserving the same depth, positions, and predictive objective, and from inference-only sampling or search, which changes how a fixed model is queried, not how its reasoning process is learned. The computational primitives introduced next reopen these three reference points in turn: recurrent processing reopens depth, intermediate-state insertion reopens sequence positions, and objective reshaping reopens the predictive objective.

2.2 Computational Primitives for Reasoning Mechanisms

Figure 2 illustrates three computational primitives that reopen the reference points fixed by the autoregressive baseline: recurrent processing, intermediate-state insertion and objective reshaping, one for each aspect the baseline fixes.

2.2.1 Recurrent Processing. Recurrent processing increases the number of transformations applied to existing positions while leaving the sequence itself unchanged. A convenient abstraction decomposes the stack into a prelude P , a block R_ϕ that may be applied more than once, and a coda C [55]. Writing $e = P(x)$ for the prelude representation,

$$h^{(t)} = R_\phi(h^{(t-1)}, e), \quad t = 1, \dots, T, \quad p_\theta(\cdot \mid x_{<i}) = \text{softmax}(WC(h^{(T)})_{i-1}), \quad (2)$$

with the standard model recovered at $T = 1$. The number of applications may also differ by position: position i freezes after T_i steps, with $h_i^{(t)} = h_i^{(t-1)}$ for $t > T_i$. Repeated application extends the effective depth, and making T_i position-dependent lets it vary across the sequence [34, 245].

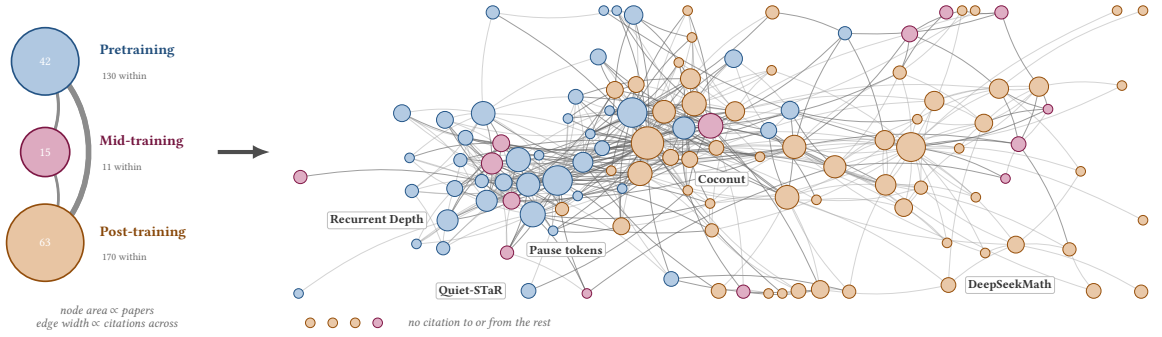


Fig. 3. Citation graph over the 120 most active papers. Colour gives the training phase, node area the within-set citations received, and thicker edges a crossing between phases.

The scored positions and prediction targets can remain unchanged; only the computational depth becomes a variable of the model. For a fixed depth T , the objective can be written as

$$\mathcal{L}_T(\theta, \phi) = - \sum_{x \in \mathcal{D}} \sum_{i \in S} \log p_{\theta, \phi}(x_i | x_{<i}; T). \quad (3)$$

When multiple depths are trained jointly, one may instead optimise an expectation over $T \sim \Lambda$. Training across multiple values of T can make the model usable at more than one depth. If computation itself is priced, an additional regulariser $\lambda \Omega(T)$ can be introduced; how such costs determine the allocated depth is a control problem taken up in §5.

2.2.2 Intermediate-State Insertion. Intermediate-state insertion augments the baseline sequence at a set of insertion sites $J \subseteq \{1, \dots, |x|\}$, with c_j additional positions introduced at each $j \in J$. For a single insertion site $j \in J$, this gives

$$\tilde{x} = [x_{1:j}, u_1, \dots, u_{c_j}, x_{j+1:|x|}]. \quad (4)$$

These positions are additional because they mediate computation, not because the input–output interface requires them. Absence from the training text is not the criterion, since reasoning traces may themselves be training targets [130, 213].

Because the sequence is now $\tilde{x}$, the training objective must specify which positions are scored. Let $S \subseteq \{1, \dots, |\tilde{x}|\}$ denote this set; training then minimises

$$\mathcal{L}(\theta) = - \sum_{x \in \mathcal{D}} \sum_{i \in S} \log p_{\theta}(\tilde{x}_i | \tilde{x}_{<i}). \quad (5)$$

Taking S to include every position gives ordinary next-token prediction over the augmented sequence, while Equation (1) is recovered when no positions are inserted. Intermediate-state insertion may be combined with objective reshaping [173]; the primitives are not mutually exclusive. §3.2 examines what the inserted positions carry.

2.2.3 Objective Reshaping. Objective reshaping changes what the model is trained to predict, encode, or preserve while leaving computational depth and sequence positions unchanged. A common form modifies the target at position i , which we denote by q_i , with ℓ the per-position loss,

$$\mathcal{L}(\theta) = \sum_{x \in \mathcal{D}} \sum_{i \in S} \ell(p_{\theta}(\cdot | x_{<i}), q_i). \quad (6)$$

Equation (1) is recovered when ℓ is cross-entropy and q_i is the observed corpus token. Common variants either replace the prediction target or add an auxiliary objective, for example $\mathcal{L}_{\text{NTP}} + \lambda \mathcal{L}_{\text{aux}}$.

When an auxiliary target is used only during training, it increases training-time complexity while leaving the deployed forward pass unchanged [57]. Internalisation methods instead use modified objectives to train the model to

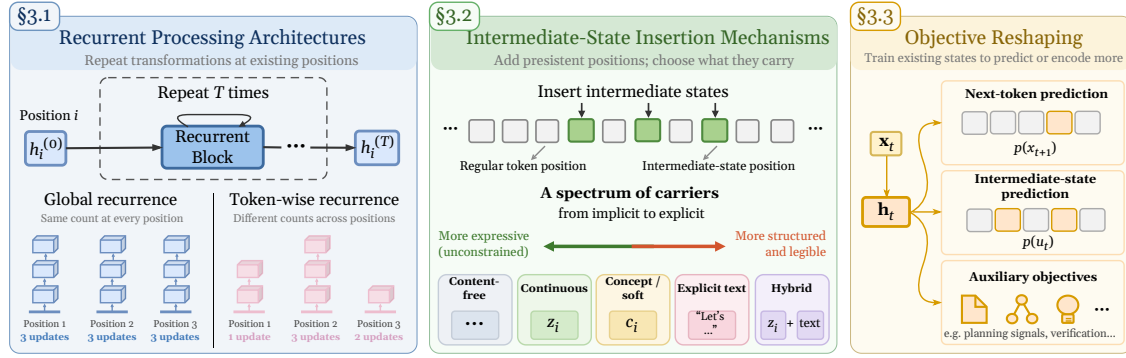


Fig. 4. Overview of mechanisms of extra training complexity.

reach an answer without explicitly generating the reasoning trajectory used for supervision [36, 37]. Together, these cases show that extra training complexity need not appear as additional deployed computation.

2.3 Supervision Availability

We distinguish training phases by the supervision a recipe can assume. *Pretraining* can begin from random initialisation and relies only on supervision from the corpus and objective themselves. *Mid-training* presupposes a trained model but still derives its signal from ordinary text or from the model itself, whereas *post-training* additionally assumes access to task-specific supervision such as teacher trajectories, checkable outcomes, process feedback, or preferences.

Figure 3 maps the 120 most active works in our corpus under this division, with 42 in pretraining, 15 in mid-training, and 63 in post-training. The citation structure shows that pretraining and post-training have developed largely as separate literatures, with most connections remaining within the same phase. Although mid-training remains much smaller, its emerging presence suggests a shift toward improving reasoning earlier and toward using ordinary corpora directly for reasoning enhancement. Training phase therefore constrains which supervision is available, but does not determine either the reasoning mechanism or the learning strategy used to acquire it. §4 organises learning by signal source: text-derived predictive supervision, self-generated and bootstrapped supervision, reasoning-trace supervision, and outcome and process feedback.

3 MECHANISMS OF EXTRA TRAINING COMPLEXITY

Extra training complexity is realised through mechanisms that alter how reasoning computation is formed. As the first of three coupled design layers, the mechanism layer asks what computational mechanism training makes available to support reasoning. Building on the three primitives introduced in §2.2, we organise these mechanisms (Figure 4) around recurrent processing (§3.1), intermediate-state insertion (§3.2), and objective reshaping (§3.3). Recurrent processing extends computation through repeated transformations, intermediate-state insertion creates persistent positions whose carriers make intermediate computation available to later steps, and objective reshaping changes what existing representations are trained to predict or encode. These mechanisms can be combined; this chapter characterises what extra training complexity makes possible, while §4 and §5 address how those mechanisms are learned and controlled.

3.1 Recurrent Processing Architectures

Recurrent processing realises extra training complexity along depth by repeatedly updating representations at existing sequence positions, following Equation (2), before anything is emitted. Universal Transformers [34], looped language models [245] and recurrent-depth models [55] instantiate the same recurrent-processing pattern, with the later two

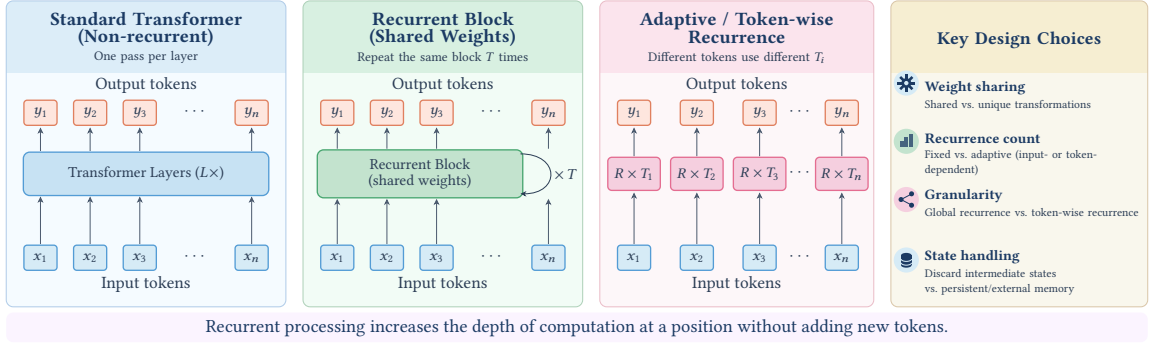


Fig. 5. Recurrence against the standard Transformer: a shared block R repeated T times globally, then T_i times token-wise, with a fourth panel of key design choices.

integrated more directly into the language-model backbone. Figure 5 sets it against a standard stack. The architectural differences between these models lie in how the recurrent transformation is parameterised, embedded into the backbone, and allowed to vary across iterations.

Recurrent-processing architectures differ primarily in how the repeated transformation is parameterised and embedded into the language model. Universal Transformer [34] shared one block of parameters across depth, buying depth without buying parameters. Such a loop can emulate iterative learning algorithms in context [207] and generalise to unseen input lengths [43]. Recurrent Depth [55] placed a recurrent core between a prelude and a coda and injected the embedded input at every step, and Ouro [245] pretrained a looped model on 7.7T tokens. PonderLM [225] instead feeds one pass’s output back into the input embedding, not an internal block. The block can be a mixture of experts [54], or made sparse to recover the efficiency that sharing costs [175]. The backbone need not be a Transformer at all: CHASE [217] looped a state-space model, and BDH-CQ [42] updated a recurrent memory from the context before iterating in latent space. Together, these designs trade parameter sharing, iteration specialisation and depth flexibility.

Scaling recurrent depth introduces a stability problem. Several recurrent models improve with additional iterations only up to a point, after which performance saturates or degrades, and STARS [208] reports that point and identifies a trade-off between stability and effectiveness. Fixed-Point Reasoners [129] attribute the collapse to a signal propagation problem that depth induces when the halting decision is postponed, and address it with pre-norm layers and residual scaling. Parcae [145] recast looping as a nonlinear time-variant system over the residual stream, locating residual explosion and loss spikes in a linear approximation to it. Two more read the loop as a dynamical system: Attractor models [46] treat the recurrence as an attractor, and Fully Looped Transformer [50] attacks the same residual explosion by routing each iteration’s output to every layer. A per-token convergence study [117] asked instead what a loop actually computes, measuring per-token convergence. Stability therefore limits how far recurrent depth can be scaled.

Recurrent processing keeps the block and the state at a fixed size while the computation they carry grows longer. The same block must therefore perform different operations at different iterations, since repeating one operation composes nothing deeper. Existing designs meet this requirement by modulating the block from one step to the next [65, 72], or by making the trajectory generative so that a deterministic recursion stops collapsing the plausible paths into one [12]. MeSH [215] traced the lag of recursive models under matched computation to both halves of this constraint: iterations computing the same thing, and one state carrying too much. Dreamer [86] identifies the constant hidden size as the bottleneck, and Adaptive Loops [48] asks whether a looped model should think harder or know more. Capacity can also be added outside the loop, through a memory buffer that preserves states standard recurrence discards [154, 215],

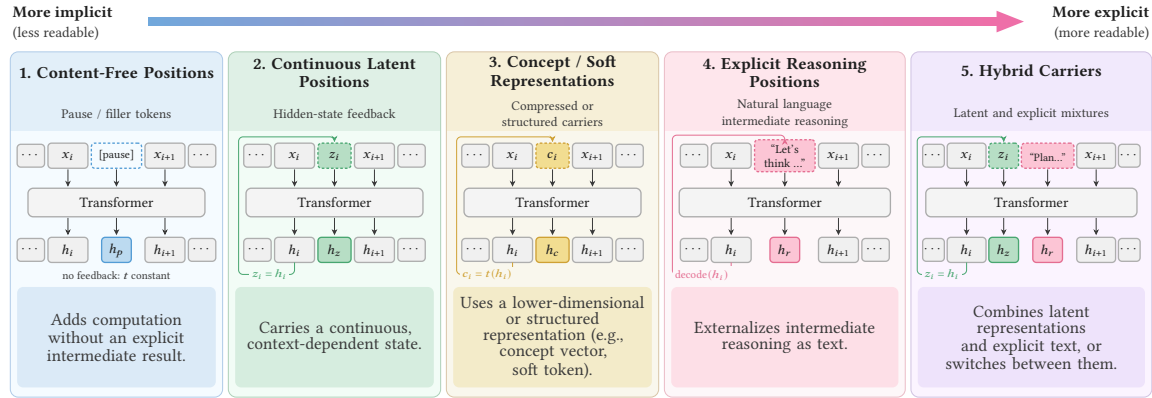


Fig. 6. Insertion carriers, five panels from content-free to hybrid along the implicit-to-explicit axis. Each loop is the carrier transform t (Table 2), absent only where t is constant.

or a looped mixture-of-experts whose pool both adds parameters and gives each pass a different operation [96]. Others repair the trajectory, rewriting each state into coordinates anchored on the last so one transition governs all [83], looping discrete embeddings [49], or supplying a latent multi-token target [161].

Recurrent processing need not be uniform: replacing a global iteration count T with a position-dependent T_i turns recurrent depth into conditional computation. One use of conditional depth is to reduce computation by skipping transformations for selected positions. MoD [150] capped the tokens that may enter a block at each layer and passed the rest along a residual connection, spending computation unevenly while fixing the total in advance. Others skip the middle layers on the ground that they are the most redundant [95]. The same degree of freedom can instead allocate additional recurrent processing to selected positions. MoR [11] routes each token to its own recursion count through a shared stack, AdaPonderLM [165] gates that repetition per token, and DNA [31] lets a token traverse any series of modules in any order. How the position-dependent depth is selected is taken up in §5.

Recurrent processing can be native to the model architecture or retrofitted into an already-trained feed-forward backbone. When recurrence is native during pretraining, the model’s representations can adapt to repeated transformation from the beginning [55, 245]. A retrofit must instead convert a trained feed-forward stack into a recurrent computation while keeping what it knows, and three conversions recur. Layer tying converts a trained model into a Recursive Transformer initialised from its own weights, so the conversion costs little performance [10]. Partial recurrence loops only the band of layers in which interpretability work locates reasoning [87], or retrofits a recurrent core into the prelude and coda layout [158]. A recurrent block can also attach to a frozen backbone [135], or run once with controllers deciding how many additive corrections to apply [69]. Recurrent processing keeps intermediate computation within depth.

3.2 Intermediate-State Insertion and Carriers

Intermediate-state insertion introduces additional positions into the sequence, while the carrier determines what representation each position makes available to subsequent computation. Following Equation (4), what an inserted position leaves behind is a transform t of the hidden state computed there, $u_{k+1} = t(h_{j+k}^{(L)})$. That transform is the *carrier*, and carrier choices in the literature form the continuum drawn in Figure 6 and summarised in Table 2. Unlike the transient iteration states of §3.1, inserted states can persist beyond the computation that created them.

Table 2. Six carrier choices for intermediate-state insertion, ordered by increasing external accessibility. t transforms the hidden state computed at an inserted position.

t	Carrier representation	External accessibility	Example
constant	a fixed vector	presence only	pause [58]
identity	the hidden state	latent tensor / probe	Coconut [60]
learned projection	a concept vector	named concept	CoCoMix [173]
soft decode	an embedding mixture	token distribution	PonderLM [225]
decode, held aside	a token on a side path	explicit text	Quiet-STaR [220]
decode	a token in the sequence	explicit reasoning text	long CoT [195]

At one extreme, the inserted symbol carries no input content and serves only to create an intermediate position, so any useful information must be created in the hidden state there. Pause-token training [58] appends learnable tokens to the input and withholds output until the last is read, with no loss at the inserted positions. The same device reappears for difficult arithmetic [68], and Catch Your Breath [51] made the count self-paced where earlier designs fixed it.

A continuous latent carrier places the result of one computation directly into the representation consumed by the next. Coconut [60] fed the last hidden state back as the next input embedding instead of decoding it. Much of what follows varies that move: latent thought vectors drawn from a prior [88, 178, 181], latent tokens that participate in computation [3, 45, 171], contemplation tokens of variable length [25, 91], latent positions replacing repeated passes [224], and a count that varies per token [223]. One objection is that a compressed thought produced one decision at a time remains token-like, answered by modelling thoughts as a Markov chain [111]. A head-free design [89] replaces the vocabulary head during reasoning, because projecting through it at every step both costs computation and forces reasoning into discrete tokens. Two placements occur: the vectors occupy positions in the emitted sequence [60, 170], or are injected into prompt placeholders before reasoning is emitted [183], with R-Capsule [156] between the two, compressing an emitted plan and feeding it back. Methods that condition ordinary hidden states without inserting a position [200] are a boundary case here. The gain is representational freedom; the cost is that the intermediate state becomes difficult to inspect without an auxiliary decoder or probe.

Between raw hidden states and words, several methods constrain the carrier into a lower-dimensional, discrete, or vocabulary-related representation. Compressing the representation, CoCoMix [173] projects the hidden state onto concepts recovered from a sparse autoencoder, so that what the position holds can be named. Next-concept prediction [116] quantised hidden states with a codebook to obtain a concept vocabulary and predicted the next concept alongside the next token; Rendered-CoT latents [19] give up continuity, arguing that a discrete latent is interpretable where a continuous one is not. Compressing the sequence, Synergy [241] routes bytes into fewer concept tokens with segmentation learned end to end, and DLCM [148] learns semantic boundaries so that computation shifts into a compressed concept space instead of being spent uniformly on locally predictable spans. Others keep the full token distribution [18].

At the explicit end of the spectrum, intermediate computation is written directly into the autoregressive sequence as ordinary language. Chain-of-thought and the long traces trained from it [32, 130, 195, 213] are the familiar case. The same carrier appears wherever a procedure can supply the text: Self-Notes [94] lets the model deviate from the input to write a note to itself and then return. Searchformer [98] and Stream of Search [52] trained from scratch on traces from A^* and from symbolic and heuristic solvers respectively, so that the model emits the search before the answer. When explicit text shares the sequence with the answer, it has three roles: it records intermediate computation, provides memory for later steps, and exposes the trajectory externally.

Hybrid carriers combine multiple representations within one reasoning trajectory. Dualformer [168] trained on search traces with parts dropped at random, giving a fast mode that emits only the solution and a slow mode that emits the reasoning. Token Assorted [169] found many words in a written chain carry no reasoning, and replaced each trace’s early portion with latent tokens, leaving the rest as text. Others blend them directly: Hybrid Latent Reasoning [219] and TARPO [232] mix one of each at every position, Soft Concept Mixing [185] mixes concept vectors into the token stream during training, Latent Thoughts Tuning [114] fuses context into latent thoughts, and HybridThinker [115] keeps compressed memory alongside the text. Hybrid carriers therefore decouple where computation happens from what must remain externally legible.

Across these carriers, none dominates in capacity, structure, persistence, and legibility. Raw latents offer the most freedom, concept carriers more structure, text the most readability at the cost of length and discretisation; hybrids combine these along one trajectory. Only one group pays a training cost to keep latent traces decodable [26, 97, 156, 237]. The carrier determines how intermediate computation is represented; §4 asks how supervision teaches its use.

3.3 Objective Reshaping and Internalisation

Objective reshaping changes internal computation by altering what representations are trained to predict or encode, while leaving sequence positions and forward depth unchanged. Equation (1) asks each position to support next-token prediction. An objective also constrains the function the learned representation must serve without uniquely specifying the computation that realises it.

One form of objective reshaping extends each position’s predictive horizon beyond the immediate next token. Multi-token prediction [57] asked each position to predict the next n tokens through n heads on a shared trunk, reporting better downstream capability. The objective becomes

$$\mathcal{L}_{\text{MTP}} = - \sum_t \sum_{k=1}^K \lambda_k \log p_{\theta}^{(k)}(x_{t+k} \mid x_{\leq t}), \quad (7)$$

with K the horizon and λ_k the weight on each offset. Larger predictive horizons can be harder to optimise, motivating curricula over K that are discussed with training-time control in §5 [8]. Predicting K exact tokens also reaches only a limited distance into the future, motivating targets that summarise a longer horizon. The heads that serve the objective need not survive deployment: a longer training horizon need not create a longer inference trajectory.

Predictive targets need not remain lexical: an objective can instead require each hidden state to approximate a structured representation of its future context. With z_t^* the target, g_{θ} the head predicting it from the state h_t , and d a discrepancy, these methods take the form

$$\mathcal{L} = \mathcal{L}_{\text{NTP}} + \lambda \sum_t d(g_{\theta}(h_t), z_t^*). \quad (8)$$

The idea generalises from future summaries [124] to semantic targets, and SONAR-LLM [39] takes z_t^* to be a sentence embedding recovered through a frozen decoder. Intermediate-state insertion (§3.2) adds a position and carrier; objective reshaping changes what an existing state encodes. The two combine in CoCoMix [173], which inserts a carrier while reshaping the objective training it.

A further form of objective reshaping internalises computation that is explicit during training into the model’s representations or parameters. A common formulation combines a task loss with a discrepancy between the student’s representation and the teacher’s. Implicit-CoT distillation [37] distils a teacher’s intermediate steps into the student’s hidden states, so that reasoning proceeds vertically across layers instead of horizontally across emitted words. Step-wise internalisation [36] removes those steps one at a time until the model answers directly, and Mutual-information

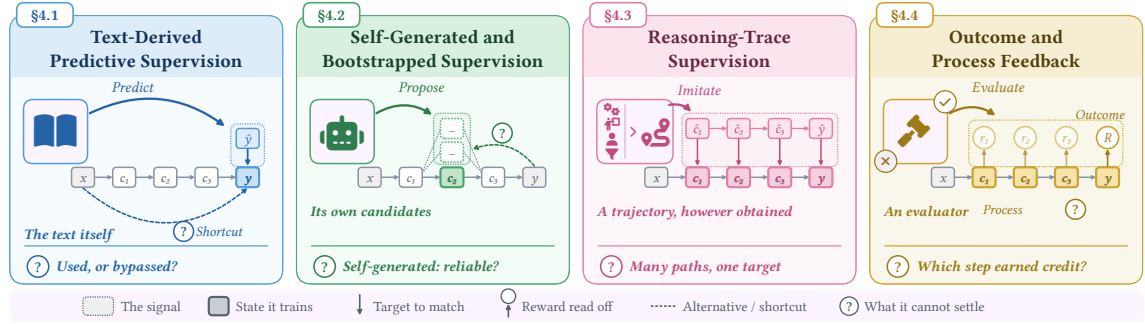


Fig. 7. Overview of learning reasoning mechanisms.

distillation [22] keeps the rationale as an auxiliary target beside the label, never as text to emit. Recurrent processing changes how many transformations are applied, intermediate-state insertion changes where intermediate states persist, and objective reshaping changes what existing states are trained to predict, encode, or preserve.

4 LEARNING REASONING MECHANISMS

Learning is the second coupled design layer of extra training complexity and asks how a reasoning mechanism is learned from available supervision. §2.3 showed that the three training phases differ in the supervision they offer, but phase alone does not determine how a mechanism is learned. The same learning strategy can recur across phases, while methods within one phase can rely on fundamentally different learning signals. We therefore organise learning according to the source of those signals (Figure 7), including text-derived predictive supervision (§4.1), self-generated and bootstrapped supervision (§4.2), reasoning-trace supervision (§4.3), and outcome and process feedback (§4.4). Figure 8 shows where each signal is read off and what its loss attaches to; Table 3 records what each leaves open. This keeps learning distinct from mechanism and training phase.

4.1 Text-Derived Predictive Supervision

At the most indirect end of the supervision spectrum, added computation is trained only through the prediction loss on observed text. The loss of Equation (1) can train recurrent depth, inserted positions, or other computation paths [34, 55, 58, 225, 245]. The computation runs

$$h_t^{(0)} \rightarrow h_t^{(1)} \rightarrow \dots \rightarrow h_t^{(K)} \rightarrow p_\theta(x_{t+1} | x_{\leq t}), \quad (9)$$

and the target lies at the path end or intermediate exits, never on a hidden state. Gradients reach those positions through predicted tokens, so the signal only shows whether extra computation improves prediction. Next-token prediction can train a new path without specifying its content.

The same corpus can provide denser supervision when each representation is trained against a broader or more abstract view of its future context. Two dimensions are available: a longer horizon and a less lexical target. Multi-token prediction [57] widens the horizon to the next several tokens (Equation (7)), and MTP Curriculum [8] moves that horizon over the course of training. The summary predicted by Future Summary Prediction [124] is itself computed from the continuation the corpus already supplies, and SONAR-LLM [39] asks each position for a sentence embedding while the gradient remains a token-level cross-entropy taken through a frozen decoder. Read in order, the targets form one progression: the next token, several future tokens, a summary of a long horizon, and a semantic representation. Along this progression, the supervision becomes richer while remaining derivable from the corpus, without an externally supplied trajectory or task reward.

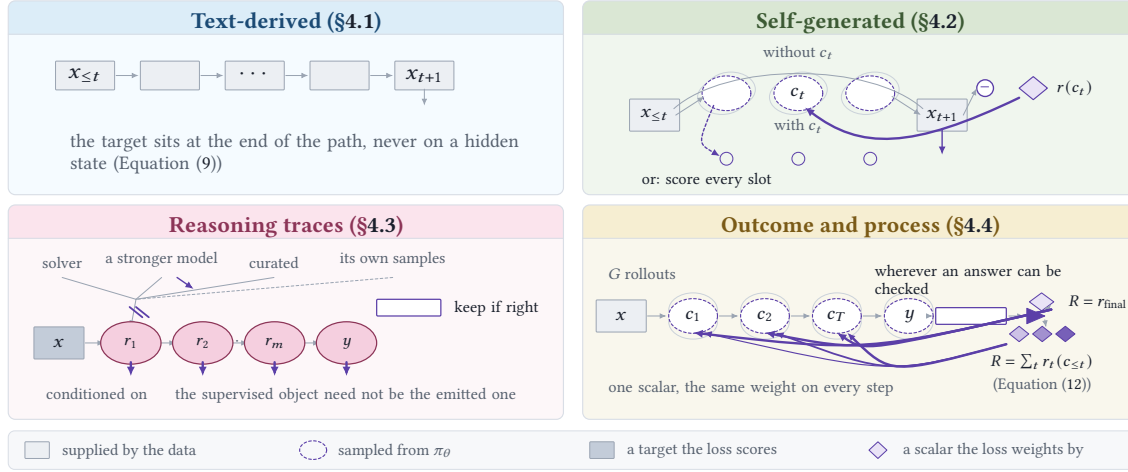


Fig. 8. One cell per learning signal (§4): where the discriminating signal is read off, and what the loss attaches to.

Text-derived supervision is not tied to training from scratch: the same signals can also install or adapt computation in an already pretrained model [10, 72, 87, 125, 137]. A pretrained checkpoint exists, its computation path is altered, and training continues on ordinary text. The defining property of this regime is the source of the target, not whether training starts from a random initialisation or a pretrained checkpoint. The central limitation is that corpus-derived targets specify what should be predicted, not how that prediction should be computed. A model given more depth or more positions may learn to use them, or may find a shortcut that lowers the same loss without using them, and an ordinary token loss cannot tell which. A lower loss identifies a computation path that helps prediction, without establishing that it reasons. A stronger signal must therefore distinguish among candidate intermediate computations, not merely score their final prediction; §4.2 turns to signals constructed from the model’s own candidates.

4.2 Self-Generated and Bootstrapped Supervision

Self-generated supervision begins when the model produces candidate intermediate computations and training distinguishes among them using signals derived from their consequences. Quiet-STaR [220] draws a rationale after every token of an ordinary corpus and scores it by how much conditioning on it raises the likelihood of the text that follows, so the computation runs $x_{\leq t} \rightarrow c \rightarrow x_t$ with c sampled from the model. The rationales are produced in parallel across positions and the gradient is estimated with REINFORCE because the tokens are sampled; Fast Quiet-STaR [71] cuts the inference overhead of generating one at every position. Self-generation alone does not define this regime; the discriminating signal must also be constructed without an externally supplied reasoning trajectory. Intermediate computation becomes an object of selection, no longer only an unobserved route through which prediction loss backpropagates.

A second line makes selection explicit by treating generated reasoning as an action rewarded from pretraining text. Methods differ in how candidates are generated and evaluated. Reinforcement Pre-Training [38] reframes next-token prediction as a reasoning problem, using the observed continuation as a corpus-derived reward, while RLPT [102] extends the target to the next segment. RLP [61] treats a sampled chain as an exploratory action and scores the information it carries about the next token,

$$r(c_t) = \log p_\theta(x_{t+1} | x_{\leq t}, c_t) - \log p_\theta(x_{t+1} | x_{\leq t}), \quad (10)$$

Table 3. The four learning-signal families of §4, separated by the source of the signal, not by training phase. *What remains unresolved* names the question each signal cannot settle.

Learning-signal family	Signal construction	Data required	What remains unresolved	Representative methods
Text-derived predictive supervision (§4.1)	targets are obtained directly from observed text or representations derived from it	unannotated text, with no reasoning traces or task labels	whether the learned representation actually supports the intended reasoning computation	[8, 10, 34, 39, 55, 57, 58, 72, 87, 124, 125, 137, 225, 245]
Self-generated and bootstrapped supervision (§4.2)	the model generates candidates, confidence estimates, or other signals from its own computation	ordinary corpora or unlabelled questions	whether the self-generated signal is reliable enough to distinguish useful computation	[1, 38, 61, 64, 71, 102, 110, 152, 176, 220, 231, 235]
Reasoning-trace supervision (§4.3)	explicit reasoning trajectories are generated or collected and used as training targets	distilled, curated, solver-generated, or self-generated traces	whether each supervised step is necessary, correct, or merely stylistic	[36, 52, 59, 98, 101, 130, 162, 168, 169, 213, 221, 228]
Outcome and process feedback (§4.4)	rollouts are evaluated through final outcomes or step-level process feedback	prompts with checkable outcomes or step-level feedback	which intermediate decisions deserve credit for the observed outcome	[14, 32, 70, 92, 103, 157, 165, 177, 187, 198, 216, 226]

which rewards a thought only insofar as it improves prediction of observed text. LoopRPT [176] varies the other axis, moving the candidate into recurrent latent computation, on the observation that a reward reaching only the emitted tokens mismatches an architecture that reasons in latent steps. Together, these methods turn ordinary text from a source of prediction targets into an environment in which the model can explore and evaluate its own computation.

Self-generated supervision becomes bootstrapping when the outputs of one round of reasoning are reused to improve the model that generates the next. Two routes recur. One retains the generated computation as supervision. BoLT [152] reads web text as the compressed outcome of a verbose thought process, infers the latent thoughts that would have produced it, and trains on them, and Semi-supervised CoT [64] samples several pseudo-chains for unlabelled questions and reuses them as a semi-supervised signal. Hidden Decoding [110] adds the same computation as latent positions where these methods add text. The other builds the judgement from the model’s own statistics, either minimising predictive entropy on unlabelled questions [1, 235] or rewarding the agreement among several trajectories it samples for the same unlabelled question [231]. Bootstrapping reduces dependence on external supervision, but the model often contributes both the candidate and the criterion that accepts it. Self-generated signals solve the availability of supervision and not its reliability. When a reasoning trajectory itself is available as a target, training can supervise intermediate computation directly, without inferring its value from the final answer.

4.3 Reasoning-Trace Supervision

Reasoning traces supervise intermediate computation directly: the datum is a full path $x \rightarrow r_1, \dots, r_m \rightarrow y$ whose intermediate states are no longer latent, and the loss scores all of them,

$$\mathcal{L}_{\text{trace}} = - \sum_{j=1}^m \log p_{\theta}(r_j \mid x, r_{<j}) - \log p_{\theta}(y \mid x, r_{1:m}). \quad (11)$$

Where the path comes from divides this literature three ways. A solver or search procedure can produce it [52, 94, 98, 168], as §3.2 describes. A stronger model can write it [101], it can be curated by hand [130, 213], or generated over text for continued pretraining [121, 186, 204]. Traces can supervise critiques, or memory and reasoning separately [77, 191]. The benefit is finer credit assignment to intermediate states; the bottleneck is finding trajectories worth imitating.

A model can generate its own candidate trajectories, filter them, and reuse the accepted ones as supervised data. STaR [221] introduced that loop, keeping a sampled rationale when it led to the right answer. The filter is the design choice: keeping every trace whose final answer is right admits paths that reach it by a wrong route. Guiding the search

with a process reward instead scores partial trajectories as they are built [59, 228]. The distinction from §4.2 is the source of the signal: there the corpus or the model’s own statistics, here a trajectory taken as a target to imitate.

A trace can supervise computation during training without remaining an explicit trajectory at deployment. One route internalises it, so that the trace is the target during training and the objectives of §3.3 dispose of it afterwards [22, 24, 36, 37]. Related work keeps the chain but shortens it, fusing token pairs into superposed embeddings [202], and one distillation objective is shaped so that the student can still be given early exits [81]. A second route relocates the trace instead of removing it, transferring its structure into a latent carrier. A curriculum replaces textual steps with latent ones [60, 171], an alignment loss ties each latent state to the corresponding step of an explicit chain [188, 196], and a compression maps m written steps onto $k \leq m$ latent positions [25, 123, 156, 164]. In both routes the choice is made once, for the whole trajectory.

Hybrid methods relax this constraint by using traces to supervise both the switch point and the step content, compressing local or easy steps while retaining selected states as explicit checkpoints [114, 115, 185, 219, 232]. Token Assorted [169] replaces the opening span of a written chain with latent tokens from a learned codebook and leaves the rest as text. The trace specifies the computational path, while the carrier determines how much remains visible at inference. The limitation is structural: trace supervision selects one trajectory to imitate even when many valid paths exist, and that trace may contain unnecessary, stylistic, or erroneous steps.

4.4 Outcome and Process Feedback

When no single reasoning trajectory should be imitated, training can instead evaluate computation by its consequences.¹ A chain c and an answer y are both sampled from the current policy, $c \sim \pi_\theta(\cdot \mid x)$ and $y \sim \pi_\theta(\cdot \mid x, c)$, and the reward is read off the answer alone as $R(c, y) = R_{\text{outcome}}(y)$. Group-relative optimisation made the scheme practical at scale [157], and DeepSeek-R1 [32] and, with a different recipe, Kimi k1.5 [177] showed that long traces emerge from outcome supervision alone with no imitation stage, which open reproductions then examined on base models [70, 216, 226]. The signal scales wherever an answer can be checked, with no annotation required. The same freedom makes it sparse: a correct answer identifies a successful trajectory, not which intermediate computations made it successful. Process supervision addresses this ambiguity by evaluating intermediate reasoning states rather than waiting until the final answer. The contrast is in where the reward is accumulated,

$$R(c_{1:T}) = r_{\text{final}} \quad \text{against} \quad R(c_{1:T}) = \sum_{t=1}^T r_t(c_{\leq t}), \quad (12)$$

and the literature moves towards step-level feedback. Math-Shepherd [187] scores each step with a process reward model trained without human annotation, and Step-DPO [92] optimises preferences at step level. Related work localises credit inside the trace, adds a flow-based process reward on top of the binary outcome, and distils a per-token signal from the model’s own successful attempts [47, 62, 80, 103, 211, 229, 238]. Finer feedback improves credit assignment, but it requires stronger assumptions about which intermediate steps deserve reward.

The same feedback principle applies whether intermediate computation is expressed in text or in latent states. The policy then emits states, not tokens, $z_{1:T} \sim \pi_\theta(z_t \mid x, z_{<t})$, and the reward $R(x, z_{1:T}, y)$ still updates whatever produced them, as latent-thought-trajectory RL [198] does by rewarding the loop steps of a recurrent model directly. Group-relative optimisation relies on the rollouts within a group differing, and a deterministic latent path collapses the group advantage to zero, which a line of work repairs by putting the randomness back in a different place [18, 35, 78, 79, 180, 242, 243]. Others address credit assignment inside the latent path instead [239], or run the reward over a sequence

¹Recent surveys cover reinforcement learning and post-training for reasoning more broadly [90, 203].

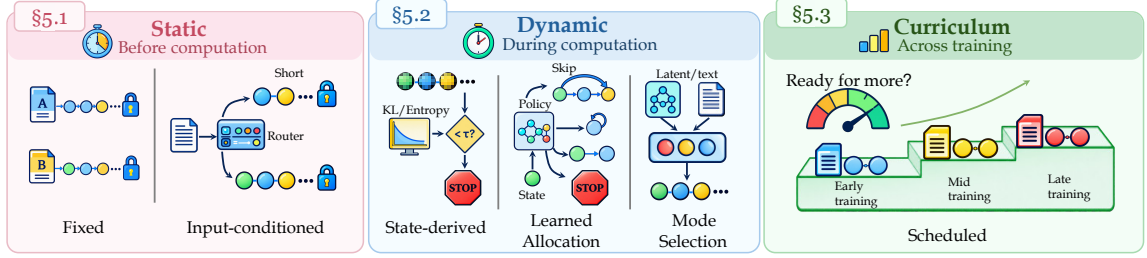


Fig. 9. Overview of control of reasoning computation.

that is part latent and part explicit [144, 219, 232]. Feedback therefore supervises a computation’s utility and not its representation, cutting across the carriers of §3.2.

Once feedback can evaluate computation, it can also decide whether another unit of computation is worth its cost. The objective acquires a price,

$$\max_{\theta} \mathbb{E}[R_{\text{task}} - \lambda C_{\text{compute}}], \quad (13)$$

so continuing or stopping becomes an action the same feedback can train. It appears in learned halting [14, 165], self-paced delay [51], skip-or-loop decisions [104], and early exit with reasoning externalised in words [140]. How much computation to perform is the allocation problem of §5.

5 CONTROL OF REASONING COMPUTATION

Control is the third coupled design layer of extra training complexity and asks how a learned reasoning mechanism is scheduled, allocated, or terminated. Different mechanisms expose different quantities to control, including recurrent depth, inserted positions, latent steps, reasoning length, and the form of the intermediate carrier. We represent these choices through an allocation policy π that trades task utility against computational cost,

$$\max_{\pi} \mathbb{E}[R_{\text{task}} - \lambda C_{\text{compute}}]. \quad (14)$$

We organise control according to when the decision is made (Figure 9). Static control fixes the allocation before computation begins (§5.1), dynamic control revises it using the evolving computation (§5.2), and curriculum control changes the available computation across training (§5.3).

5.1 Static Control Before Computation

The simplest allocation policy assigns the same preset computation budget to every input. The constant $B(x) = B_0$ is set by whoever trains the model, before any input is seen. Families differ in what the constant counts: delay tokens, latent thoughts or prediction heads [57, 58, 60], loop iterations or forced trace length [10, 130], yet the decision rule is identical. None of these methods claims its constant is right for a particular input. Fixed budgets therefore establish the value of additional computation without solving the allocation problem itself.

A more selective strategy chooses the budget from properties of the input before reasoning begins. The budget becomes a function of the input, $B(x) = g(x)$ if hand-designed and $B(x) = g_{\phi}(x)$ if fitted, giving easy and hard inputs different amounts of computation. Recomputable rules are rare here: one reads difficulty from a running per-question accuracy estimate [122]. Most instances of g are fitted, on query features [40] or on a choice between a short and a long reasoning mode before decoding [27, 236]. Input difficulty is only a proxy for the computation an input will need, and the budget is committed before any of it has been observed. This limitation motivates allocation rules that revise the budget after observing the computation itself.

5.2 Dynamic Control During Computation

Once computation has begun, intermediate states can provide direct evidence of whether another step is still changing the model’s prediction. The rule is a convergence test on the model’s own outputs: halt when $d(s_t, s_{t-1}) < \epsilon$, concretely $D_{\text{KL}}(p_t \| p_{t-1}) < \epsilon$ on the output distributions the states already emit [55, 117]. No separate allocator is trained: the signal sits in states already computed, and the decision is reproducible. On the text axis the reproducible signal is entropy over the emitted trace [106, 240], and one that looks reproducible instead scores the sufficiency of what has been written [184]. Observable change is not task utility: a step can move the distribution without improving the answer, and a settled state can still be wrong, so ϵ and d remain hand-set.

Instead of relying on a hand-designed stopping rule, a learned controller can allocate computation from the evolving internal state. Such controllers differ in the level at which they act, not in their architecture: $a_t \sim \pi_\phi(a_t | x, s_t)$ over $\{\text{continue, halt, skip, repeat}\}$ lets one policy spend more, less, or stop. Below the input-level routers chosen before computation begins, the decision attaches to a single token [11, 150, 165, 245], to each layer inside one token’s pass [13, 93, 128], or to where an inserted delay belongs [51, 84], with one line separating where the budget goes from when it stops [197]. Coarser still, a latent trace halts after a learned number of refinement steps [69, 131], and a cost-carrying reward sets a reasoning trace’s length [2, 201, 214, 230]. A fitted mapping reads more of the state than a divergence or entropy threshold admits. The price is supervision or reward for the allocator itself (§4.4), and a decision that survives only in the weights, where no reader can reproduce it. In every case the action is an amount.

Adaptive allocation can control not only how much computation is performed, but also which computational mode receives the next unit of budget. The action set widens to

$$a_t \in \{\text{loop, skip, latent, text, halt}\}, \quad (15)$$

where each continuing action names a mechanism of §3 and halting ends the trajectory, so the decision is no longer a scalar amount on one axis. Controllers switch between fast and slow reasoning modes [27, 236], and decide at each step whether the next state is written or latent and when to leave that trace [109, 131]. Form is itself an action [104, 140]. One price C_{compute} in Equation (13) covers both, counted in emitted tokens, depth, latency, or latent operations, so how much and what kind are learned jointly. Allocation therefore becomes a sequential control problem over both the quantity and the form of computation.

5.3 Curriculum Control Across Training

The computation budget itself can also vary over training, so the model meets progressively different depths, horizons, or reasoning lengths. Training fixes a sequence $B_1, \dots, B_E$ or a function $B_e = f(e)$ of training progress, while a stationary distribution resampled every step [55] spans the same budgets without ordering them. The varying quantity is the effective depth [125, 137], the prediction horizon [8], or the number of latent steps [60, 171]. Each is a compute budget, not training-data difficulty, which separates these from task curricula [136]. What the schedule buys is easier optimisation: the model learns a short computation before it is asked for a long one [17].

Most existing computation curricula schedule these changes by training step, not by whether the model is ready to use a larger budget. A schedule $B_e = f(e)$ settles the trajectory when the run is configured [8, 125]. Training step is only a proxy for readiness, and a rule $B_e = f(m_e)$, with m_e a maturity signal, could use convergence, representation quality or reasoning success. Defining and measuring such a signal is the problem §6.4 takes up. Learning when a model is ready for more computation remains open and reappears as a central question in §7.

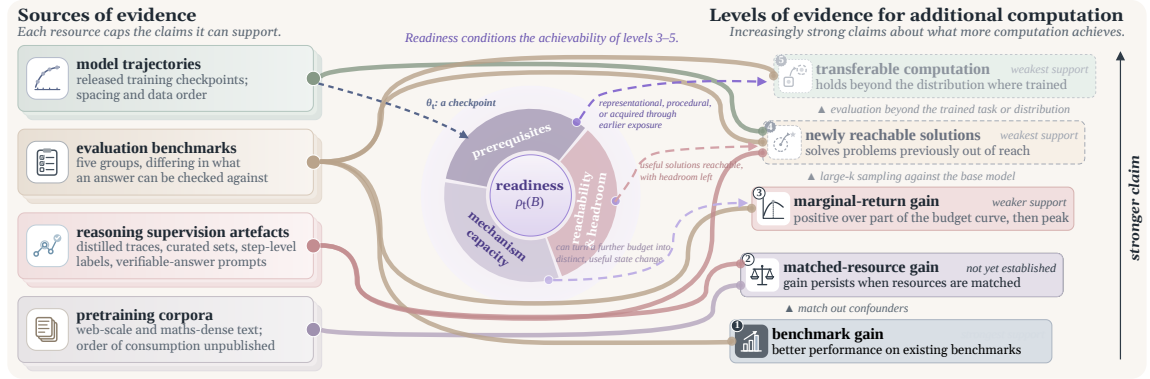


Fig. 10. Overview of empirical foundations and evidence.

Table 4. Open model families releasing intermediate checkpoints. Token interval sets the resolution for locating a change during training; no column records the order of the training data.

Model	Scale	Training tokens	Checkpoints	Token interval
Pythia [16]	70M–12B	300B	154	2.1B
OLMo 2 [133]	1B–32B	4.1–6.3T	268–965	4.2–21B
Olmo 3 [132]	7B, 32B	6.1T	1,486	4.2B
SmolLM2 [5]	135M–1.7B	11T	41	270B
TinyLlama [233]	1.1B	3T	286	10.5B
Huginn [55]	3.5B	800B	8 of ~350	100B

Table 7 sets the three preceding chapters side by side, giving each family its representative methods and the value the survey assigns it on every axis. The blanks are informative: control is stated for about half the families, and the explicit-text carrier divides into three families that share a mechanism and differ only in the signal that trains it.

6 EMPIRICAL FOUNDATIONS AND EVIDENCE

Having organised reasoning enhancement by mechanism, learning, and control, we now ask what empirical resources and evidence support claims about these designs (Figure 10). We first survey model trajectories (§6.1), training corpora, supervision artefacts, and benchmarks (§6.2) that make longitudinal and controlled analysis possible. We then examine what current experiments establish (§6.3), separating benchmark gains from stronger evidence based on matched resources, attribution to a single layer, newly reachable reasoning, and transferable capability. Finally, we ask when a model state is ready for a design (§6.4), gathering the evidence on prerequisite competence, reachable headroom, and compatibility.

6.1 Models and Training Trajectories

Locating when a reasoning mechanism emerges requires models that release training trajectories, not only final weights. Table 4 lists six releases. Pythia [16] and the two OLMo generations [132, 133] also publish what each checkpoint had already consumed, which makes them the ordinary language models best suited to training-dynamics analysis. TinyLlama [233] publishes no order, so a transition found in it can be placed in training time, not in the corpus. SmolLM2 [5] publishes no order either, and keeps them in a repository its model card does not name. Huginn [55] is the only one whose pretraining installs a recurrent processing mechanism, and publishes no order.

The limitation is therefore not model scale but the scarcity of dense trajectories for models that contain such a pathway. Huginn’s eight points fix the resolution at which a change in its use can be seen. The mechanism is not

Table 5. Corpora and reasoning supervision sets the surveyed methods train on. Counts are tokens for corpora, examples for trace sets, step annotations where marked, and problems for problem sets.

Category	Dataset
General training corpora	Pile [53] (300B), C4 [149] (156B), RedPajama-1T [194] (1.2T), SlimPajama [160] (627B), Dolma [163] (3T), FineWeb [141] (15T) and FineWeb-Edu (1.3T), Nemotron-CC [167] (6.3T), DCLM-Baseline [100] (1.35T), OpenWebMath [138] (14.7B), Proof-Pile-2 [9] (55B), MathPile [192] (9.5B)
Reasoning traces and labels	OpenThoughts-114k (114k), Bespoke-Stratos-17k (17k), s1K [130] (1,000), LIMO [213] (817), PRM800K [107] (800k steps), Math-Shepherd [187] (400k steps)
Problem sets	OpenR1-Math-220k (220k), NuminaMath-CoT (860k), OpenMathReasoning (306k), DAPO-Math-17k [216] (17k), DeepScaleR-Preview (40k)

Table 6. Benchmarks the surveyed methods report on, grouped by subject area. The groups differ in what an answer can be checked against: tests or a final answer in mathematics and code.

Category	Dataset and benchmark
Mathematics	GSM8K [29], MATH [67] and its MATH-500 subset [107], AIME 2024/2025, AMC23, OlympiadBench [63], MinervaMath [99], SVAMP [139], ASDiv [127], TheoremQA [21]
Code	MBPP [7], LiveCodeBench [73], SWE-bench [75] and its Verified subset, CodeContests
General knowledge	MMLU [66], MMLU-Pro [190], GPQA and GPQA-Diamond [151], ARC-Challenge [28], HellaSwag [222], CommonsenseQA [174], StrategyQA [56], DROP [41], HotpotQA [210], MuSR [166], HLE [143]
Logic and algorithmic	BBH [172], ProntoQA [153], ProsQA [60], LogiQA, ZebraLogic [108], Countdown [52], Game of 24 [212], Sudoku-Extreme and Maze-Hard [182], parity [85], Blocksworld [179]
Domain-specific	MedQA [76], ScienceQA [119]

confined to research models, since DeepSeek-V3 [33] trains with a multi-token prediction objective, but no production run of that kind releases a trajectory. The corpora behind these runs carry their own constraints.

6.2 Data and Evaluation Resources

General pretraining corpora. The corpora used by this literature are the ordinary ones, listed in Table 5. Mid-training methods are trained almost entirely on them, since their stage definition requires a signal computable from ordinary text. Two are no longer available as published: the Pile’s original host withdrew it, and the Books subset of RedPajama-1T was disabled in 2024.

Reasoning supervision artefacts. Reasoning supervision comes in four kinds, also in Table 5: distilled trace sets, curated sets of roughly a thousand examples, step-level labels, and prompt sets with checkable answers. A path, a step and an outcome are three different objects, so a set built for one cannot directly train a method fitted to another.

Evaluation benchmarks. Mathematics, code and general reasoning carry most of the reported numbers, and Table 6 compiles what the surveyed methods report on. More reasoning-dense benchmarks beyond these areas would widen what this literature can measure.

6.3 What Current Evidence Establishes

A benchmark improvement establishes that a complete recipe beats its baseline, but it does not identify which part of the extra-training-complexity design produced the gain. Dot-by-dot filler tokens carry no semantic content, yet they let a model solve two algorithmic tasks it cannot solve without intermediate tokens [142]. A recurrent-depth model improves on further passes while emitting no additional tokens [55]. Both hold the added content fixed, yet each sits inside a recipe that moves mechanism, learning signal, and control as a set. Benchmark gain is therefore evidence that a recipe works, and not yet evidence for the causal value of the design choice it contains.

A stronger claim requires controls that match the other resources an intervention changes, because an extra-training-complexity design rarely moves one alone. Increasing recurrent depth also changes parameter sharing and state reuse, inserted positions enlarge what later tokens attend over, an auxiliary objective changes supervision, and a latent carrier

Table 7. Representative method families on the three coupled design layers. Chips abbreviate the categories of §§3–5, two in a cell where both apply; a dash marks a value the survey does not state.

Method family	Representative methods	Mechanism	Learning	Control	Main limitation
<i>Recurrent processing: repeated transformations extend computation at existing positions</i>					
Shared recurrence	Universal Transformer, Recurrent Depth, Ouro [34, 54, 55, 175, 245]	Recurrent	Text	Dynamic	useful iterations cannot always be increased indefinitely
Conditional recurrence	MoR, AdaPonderLM, DNA [11, 31, 95, 150, 165]	Recurrent	Feedback	Dynamic	the decision survives only in the weights, where no reader can reproduce it
Retrofitted recurrence	Recursive Transformer, ETD [10, 69, 87, 135, 158]	Recurrent	Text	Static Curriculum	the conversion must keep what the feed-forward stack already knows
<i>Intermediate-state insertion: persistent positions make intermediate computation available to later steps</i>					
Content-free positions	Pause tokens, Catch Your Breath [51, 58, 68]	Insertion	Text Feedback	Static Dynamic	almost nothing about the state computed there is recoverable
Ordinary text as a scoring environment	Quiet-STaR, Reinforcement Pre-Training, RLP [38, 61, 71, 102, 176, 220]	Insertion Objective	Self	–	the model supplies both the candidate and the criterion that accepts it
Continuous latent carriers	Coconut, R-Capsule [25, 60, 88, 89, 156, 170, 171, 183, 224]	Insertion	Trace	Curriculum	the intermediate state resists inspection without a decoder or probe
Solver-supplied and curated traces	Searchformer, Stream of Search, s1 [52, 94, 98, 101, 130, 204, 213]	Insertion	Trace	Static	one trajectory must be selected to imitate where reasoning admits many
Self-taught explicit traces	STaR, ReST-EM, rStar-Math [59, 162, 221, 228]	Insertion	Trace	–	an answer-level filter admits right answers reached by a wrong route
Hybrid latent and textual carriers	Dualformer, Token Assorted [115, 168, 169, 219, 232]	Insertion	Trace	–	the boundary between carriers is learned indirectly, from reward or a preset action set
Traces trained by outcome feedback	DeepSeek-R1, Kimi k1.5 [32, 70, 157, 177, 216, 226]	Insertion	Feedback	–	a correct answer identifies a successful trajectory, not the steps that earned it
<i>Objective reshaping: training changes what existing representations must predict or encode</i>					
Broadened predictive horizon	Multi-token prediction, MTP Curriculum [8, 57]	Objective	Text	Static Curriculum	corpus targets specify what should be predicted, not how it should be computed
Structured auxiliary targets	Future Summary Prediction, SONAR-LLM [39, 124, 173]	Objective	Text	–	the objective fixes what a state must serve, not the computation realising it
Internalisation of an explicit trace	Implicit-CoT distillation, Stepwise internalisation [22, 24, 36, 37]	Objective	Trace	–	the deployed model no longer performs the reasoning it was trained from

changes representational bandwidth (§§3.1–3.3), alongside the extra computation. Multi-token prediction [57] runs the other way, improving code generation and small algorithmic tasks without increasing deployed computation, because the change is in the training objective alone. Probing a recursive model under matched computation locates the bottleneck in its fixed-width state [215]. A looped model measured against iso-FLOP and parameter-matched baselines still needs a memory bank [48]. A matched-resource gain therefore shows that the design contributes beyond the extra resources it changes, and it still does not isolate which design decision is responsible.

Resource matching still evaluates a design as a package, and stronger evidence attributes the gain to the mechanism, the learning signal, the control decision, or their interaction. Filler tokens pay off only under specific and dense supervision [142], and pause tokens yield gains only when the delay is present in both pretraining and finetuning [58]. An available mechanism is not a learned mechanism. A looped model still needs a memory bank for tasks that looping alone does not carry [48], so the gain cannot be assigned to recurrence alone. Current studies usually demonstrate that a complete design works, and fewer isolate what each layer contributes or how they interact. Design-attributed gain thus identifies why a recipe works within the three layers, and still does not show the model gained reasoning previously out of reach.

A still stronger claim is that an extra-training-complexity design makes reasoning strategies reachable that the base model could not reliably access, and not merely more probable among behaviours already present. Improvements in pass@1 leave the two readings indistinguishable. Large- k sampling against the base model offers one way to tell them apart in reinforcement-learning settings. A trained model leads at small k while the base model catches up or overtakes at large k [218]. Token-level entropy can rise while sampled paths converge onto fewer distinct answers [199], so apparent exploration need not mean an enlarged strategy set. Prolonged training supports the opposite reading, reaching solutions that extensive base-model sampling does not recover, with the gain correlated with the base model’s competence on the task [113]. The two readings need not conflict, since they may describe different model states, tasks or training regimes. Current evidence therefore suggests that extra training complexity can sometimes enlarge the reachable reasoning set, and the claim depends strongly on the model state and the regime being studied. The strongest claim is that the reasoning capability transfers beyond the task, the distribution, or the training configuration in which the design was learned. Changing the prompt on the same benchmark does not count. The five claims form a ladder of increasing strength,

$$\begin{aligned} \text{benchmark gain} &< \text{matched-resource gain} < \text{design-attributed gain} \\ &< \text{newly reachable reasoning} < \text{transferable reasoning capability}, \end{aligned} \quad (16)$$

where each rung requires ruling out a stronger alternative explanation than the rung below it. Benchmark and matched-resource evidence are common, design attribution is rarer, evidence for newly reachable reasoning is mixed, and evidence for transferable reasoning capability is weakest. Whether any of these claims holds is not a fixed property of a design, and depends on the model state the design is applied to (§6.4). The strength of evidence therefore depends not only on experimental controls, but also on whether the model state is one in which the design can use the structure, supervision or control it introduces.

6.4 Design Readiness and Model State

The effectiveness of an extra-training-complexity design depends on the state of the model to which it is applied, and not on training stage, parameter scale, or design choice alone. A stage label classifies a recipe by the supervision it requires and says nothing about how mature the model is, parameter scale bounds capacity without showing that a design can use it, and training step counts optimisation, not what the model has become. Strong evidence for a design in one setting does not carry to another model state (§6.3). For a mechanism with an explicit budget, readiness takes a local form, the marginal return to computation at a given model state,

$$\rho_t(B) = \frac{R(\theta_t, B + \Delta B) - R(\theta_t, B)}{\Delta C}, \quad (17)$$

with θ_t a checkpoint, B the budget it is given, ΔB the increment, R the task performance that results and ΔC the cost of that increment. A positive ρ_t is local readiness at that mechanism and that budget, and not a universal maturity. Objective reshaping and learning-signal changes need not be describable as extra deployed computation, so ρ_t does not cover every design. Readiness in general is the capacity of a model state to use a particular extra-training-complexity design, and three conditions on that capacity recur across the evidence.

The first condition is that the model has acquired the prerequisites on which the proposed design must operate. A prerequisite can be representational, procedural, acquired by earlier exposure, or plain competence at the task. The delay positions of §3.2 buy nothing when inserted at finetuning alone. They pay only when pretraining contained them as well [58], so their use must be learned during pretraining. Scale sets a further prerequisite. A multi-token objective becomes useful only once the model is large enough [57], so one modification acts differently at different model states. Earlier competence also governs how far later training gets. A study following one model from pretraining

through post-training predicts its post-reinforcement performance at fixed reinforcement compute from the preceding pretraining loss [159]. Design availability is not design usability.

Prerequisites are not sufficient, since useful reasoning behaviours must also be reachable from the current model while enough headroom remains for the design to improve them. When successful trajectories are nearly unreachable, supervision, reinforcement learning and search collect little, leaving the design little to act on. When the model already solves the task reliably, headroom is small and the design can only sharpen behaviour already present. The productive middle keeps useful behaviour reachable but not yet reliable, giving the design something to amplify or reshape. A controlled staged-training study reports capability gains only where pretraining leaves headroom and the reinforcement data target the model’s edge of competence [227]. Within one run, reinforcement learning amplifies moves an imitation policy prefers on easy problems and surfaces near-absent moves on hard ones [159]. One regime looks like reweighting and another produces newly reachable reasoning, so the opposed large- k readings of §6.3 can both stand at different starting model states. Training for longer does not monotonically improve readiness. Reinforcement learning already helps intermediate pretraining checkpoints, and supervised and mid-training data shift which trajectories feedback reaches [4, 15, 193].

Even with prerequisites and remaining headroom, improvement depends on whether the chosen mechanism, learning signal, and control strategy are compatible with the current model state. A compatible mechanism still produces task-relevant change. Recurrence can destabilise, a loop can reach a fixed point, and added positions can stop changing useful state [117, 184, 208]. Supervision has to teach or refine that mechanism. Filler tokens pay only under dense supervision, and pause positions only when pretraining and finetuning both carry the delay [58, 142], so a mechanism can exist and stay unused. Control has to invoke the computation, state or intervention at the right place, time or difficulty. A fixed allocation wastes a useful mechanism. Readiness is a property of the model and the design together, not of the model or the mechanism alone. The evidence decomposes readiness into

$$m_t = (m_t^{\text{prereq}}, m_t^{\text{reach}}, m_t^{\text{design}}), \quad (18)$$

the prerequisite competence acquired, the trajectories reachable with headroom left, and the compatibility of that state with the coupled design. This decomposition is a synthesis of the current evidence and not an established metric, and no single signal yet predicts across tasks and designs when extra training complexity will become effective.

7 VISION

Mechanism, learning, and control organise extra training complexity into three coupled layers, and §§3–5 built that organisation from methods proposed one at a time. §6 then asked what the evidence establishes, and when a model state is ready for a given design. Attribution varies the layers one at a time and design chooses them together, so the two are separate activities. This chapter takes up the second. Four directions carry it: coordinating learning signals across training stages (§7.1), adapting control across reasoning mechanisms (§7.2), placing the latent–explicit interface (§7.3), and co-designing capability (§7.4). Progress on reasoning capability will require designing the three layers jointly, and evidence that the joint design is what produced the gain.

7.1 Coordinating Learning Across Training Stages

Different training stages expose the model to different supervision (§4), suggesting complementary roles in learning to reason, not interchangeable occasions to train the same mechanism. Capabilities learned at one stage condition what later stages can teach, so the decision is how to coordinate signals across stages as the model state evolves, not which stage is best.

These stages are path-dependent, because the effect of a later signal depends on the model state produced by earlier training. Prior exposure changes whether later reinforcement learning can compose or discover useful behaviour [227]; the mid-training mixture alters the return to that reinforcement even when the optimisation procedure is unchanged [4, 193]. The sharpening reported for reinforcement learning appears where it follows supervised fine-tuning, not where it acts on a base checkpoint [15]. This path dependence suggests that training stages need not be commutative. The design object is therefore the trajectory of capabilities and signals, not one stage.

The training lifecycle then works as an adaptive policy for coordinating the acquisition and refinement of reasoning mechanisms across stages. One plausible trajectory has pretraining install a computational substrate, mid-training shape candidate internal computations, and post-training assign task-level utility to them. The current evidence permits that trajectory and does not establish it. Such a policy changes the trigger. The transition from one signal to the next would follow what the model has already learned, not only a predetermined schedule, which is the readiness question of §6.4. The broader challenge is to coordinate learning signals across the training trajectory, letting readiness, and not a lifecycle label, decide when the signal should change. A model that has learned a useful reasoning mechanism still has to decide when running it is worth its cost.

7.2 Adaptive Control Across Reasoning Mechanisms

Current allocation methods expose a tension between interpretability and adaptivity. Fixed budgets and state-derived thresholds publish their decision rule and are easy to reproduce. §5 finds that their signals report only that a state has changed or stabilised, never that another step will help. Learned routers condition on richer internal state, and their allocation criterion is buried in the controller weights, hard to inspect or reproduce. Existing methods therefore supply an observable proxy for computational need or an adaptive but opaque policy, seldom both. The open question is how control adapts to model state, task state, and the mechanisms available.

A useful abstraction is to treat allocation as estimating the marginal utility of the next computational operation relative to halting,

$$\Delta U(a_t | s_t) = \mathbb{E}[R | s_t, a_t] - \mathbb{E}[R | s_t, \text{halt}] - \lambda C(a_t), \quad (19)$$

with s_t the state, a_t a candidate operation and C its cost. The formulation accommodates results in which further computation becomes redundant or harmful once the state already suffices [184]. It also drops the assumption that allocation is one-dimensional, since a_t ranges over another loop, a latent step, an explicit token, an externalised checkpoint and termination. The typed controllers of §5 already choose among modes and not only among lengths [109, 131]. Allocation is therefore better viewed as sequential control over heterogeneous computational operations.

The unresolved challenge is to make this marginal-value estimate learnable and observable across reasoning mechanisms, and not over budget alone. Two properties matter. An allocator should be accountable, so that its decision to continue corresponds to a measurable expected improvement relative to cost and can be told apart from proxies such as uncertainty, entropy or mere state movement. It should also price heterogeneous mechanisms, since a loop, a latent step and an emitted token differ in latency, memory and representational bandwidth, and a count of tokens or layers values none of them correctly. Both must hold across difficulty and model maturity. Once the next operation can be chosen, its representation becomes part of that decision.

7.3 The Latent–Explicit Interface

The latent–explicit trade-off depends jointly on the architecture that propagates the state and on the supervision that shapes it, and it is not a property of the carrier alone. A looped model supervised with a gold reasoning step at every

latent position closes the gap to explicit chains at three billion parameters, and projecting its post-loop latents through the language head recovers those steps [44]. Recurrence without that supervision reads the other way. Logit-lens and coda-lens probes of a depth-recurrent model find limited evidence of an interpretable latent chain, and its gains fall short of externalising the steps [120]. Capacity, legibility and usefulness are therefore set jointly by carrier, architecture and supervision, so the carrier is one part of a coupled design.

Moving reasoning into a continuous space removes the vocabulary bottleneck, and it also removes the supervision and control interface that language provides. A written step has a direct training target or can be compared against a teacher trace, and it can be verified, edited or corrected externally. A latent state needs an alignment, reconstruction or outcome signal instead (§4.3), and outcome feedback is harder to assign to it because the observed answer mixes that state’s quality with later sampling noise [239]. Exploration is harder as well, since a deterministic continuous trajectory gives a group-relative update nothing to compare, the difficulty the repairs of §4.4 answer. Explicit language carries three roles at once. It is a representation at the mechanism layer, a supervision interface at the learning layer, and a control interface for correction. The linguistic bottleneck therefore constrains all three layers, not representation alone.

These trade-offs make a single carrier unlikely to be optimal across a reasoning trajectory. High-bandwidth latent states may serve local refinement and exploration, while selected states may be worth discretising or verbalising for verification, correction, communication or persistent memory, so that a trajectory alternates between private computation and explicit checkpoints. The hybrid methods of §3.2 already make the carrier a local decision, learning that boundary indirectly from task reward or a predefined action set. A more general system would condition externalisation on the role the explicit state is to play, and treat the carrier as a model decision, not a fixed architectural choice. The open problem is to learn the boundary between private computation and explicit interface, and not to choose latent or explicit globally. A more flexible representation still leaves open whether what flows through it is better reasoning.

7.4 Co-Designing Reasoning Capability

Reasoning methods now decompose cleanly into mechanism, learning, and control. Most work still settles the layers one after another. The usual sequence invents a mechanism, attaches supervision, adds a controller, then reports the assembled recipe as a single result. That sequence is a remnant of ad hoc design, each layer introduced to repair a limitation the previous one left open.

The three layers are strongly coupled. A mechanism is worth something only when training can teach the model to use it. A learning signal is effective only when the mechanism exposes states and actions it can act on. A controller is meaningful only when it can observe what the model has computed and invoke an operation that changes it. These three layers are not a sequential assembly running from mechanism to learning to control. Each constrains what the others can achieve, so they have to be designed together.

Readiness constrains the design further, since the same coupled design pays at one model state and not at another (§6.4). The useful question is therefore no longer which method is universally best. It is which mechanism, which learning signal and which control policy suit this model state and this task. Readiness also moves as training proceeds, so the design that pays at one checkpoint need not pay at the next. Design should adapt to readiness, and the three layers should be chosen for the model and the problem in front of them.

Extra training complexity is a design paradigm, and not the goal. Reasoning capability is the goal: capability that is useful, controllable, generalisable, transferable, and empirically attributable, with the evidence ladder of §6.3 fixing what such a claim must survive. Training complexity and deployed computation are costs paid towards that capability.

A design that raises either without moving capability has bought nothing. The goal is therefore not to maximise training complexity or deployed computation, but to co-design mechanisms, learning signals, and control policies that turn that complexity into demonstrable reasoning capability.

8 CONCLUSION

This survey brought together methods that add extra training complexity to improve reasoning, despite their development across largely separate literatures. We organised them by mechanism, learning, and control: what creates computational capacity, what shapes its use, and what decides when to invoke it. Viewed jointly, these layers reveal that similar mechanisms can be trained under very different supervision and that similar signals can support different mechanisms. Current evidence reaches benchmark and matched-resource gains routinely, attributes them to a single design layer far less often, and is weakest on newly reachable reasoning and transferable reasoning capability. Whether a design pays depends on the competence the model already has, on the headroom that remains reachable, and on whether the design suits that state. The central challenge is therefore to design mechanism, learning, and control together, and to show which part of that design expanded what a model can reason through.

REFERENCES

- [1] Shivam Agarwal, Zimin Zhang, Lifan Yuan, Jiawei Han, and Hao Peng. 2025. The Unreasonable Effectiveness of Entropy Minimization in LLM Reasoning. [arXiv:2505.15134](#) [cs.CL]
- [2] Pranjal Aggarwal and Sean Welleck. 2025. L1: Controlling How Long A Reasoning Model Thinks With Reinforcement Learning. [arXiv:2503.04697](#) [cs.CL]
- [3] Lukas Aichberger and Sepp Hochreiter. 2026. Unlocking the Working Memory of Large Language Models for Latent Reasoning. [arXiv:2605.30343](#) [cs.CL]
- [4] Syeda Nahida Akter, Shrimai Prabhumoye, Eric Nyberg, Mostofa Patwary, Mohammad Shoyebi, Yejin Choi, and Bryan Catanzaro. 2025. Front-Loading Reasoning: The Synergy between Pretraining and Post-Training Data. [arXiv:2510.03264](#) [cs.CL]
- [5] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydliček, Agustín Piqueres Lajarin, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, et al. 2025. SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model. [arXiv preprint arXiv:2502.02737](#). [arXiv:2502.02737](#) [cs.CL]
- [6] Mohammad Ali Alomrani, Yingxue Zhang, Derek Li, Qianyi Sun, Soumyasundar Pal, Zhanguang Zhang, Yaochen Hu, Rohan Deepak Ajwani, Antonios Valkanas, Raika Karimi, Peng Cheng, Yunzhou Wang, et al. 2025. Reasoning on a Budget: A Survey of Adaptive and Controllable Test-Time Compute in LLMs. [arXiv preprint arXiv:2507.02076](#). [arXiv:2507.02076](#) [cs.AI]
- [7] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. [arXiv:2108.07732](#) [cs.CL]
- [8] Ansar Aynedinov and Alan Akbik. 2025. Pre-Training Curriculum for Multi-Token Prediction in Language Models. [arXiv:2505.22757](#) [cs.CL]
- [9] Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An Open Language Model For Mathematics. [arXiv preprint arXiv:2310.10631](#). [arXiv:2310.10631](#) [cs.CL]
- [10] Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. 2024. Relaxed Recursive Transformers: Effective Parameter Sharing with Layer-wise LoRA. [arXiv:2410.20672](#) [cs.CL]
- [11] Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyouon Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. 2025. Mixture-of-Recursions: Learning Dynamic Recursive Depths for Adaptive Token-Level Computation. [arXiv:2507.10524](#) [cs.LG]
- [12] Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. 2026. Generative Recursive Reasoning. [arXiv:2605.19376](#) [cs.LG]
- [13] Andrii Balashov and Olena Ponomarova. 2026. TriRoute: Unified Learned Routing for Joint Adaptive Attention, Experts, and KV-Cache Allocation. [arXiv:2607.06601](#) [cs.LG]
- [14] Andrea Banino, Jan Balaguer, and Charles Blundell. 2021. PonderNet: Learning to Ponder. [arXiv:2107.05407](#) [cs.LG]
- [15] Rachit Bansal, Clara Mohri, Tian Qin, David Alvarez-Melis, and Sham Kakade. 2026. RL Excursions during Pre-Training: Re-examining Policy Optimization for LLM training. [arXiv:2606.04272](#) [cs.CL]
- [16] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O’Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, others, and Oskar van der Wal. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. [arXiv preprint arXiv:2304.01373](#). [arXiv:2304.01373](#) [cs.CL]
- [17] Dake Bu, Wei Huang, Andi Han, Atsushi Nitanda, Hau-San Wong, Qingfu Zhang, and Taiji Suzuki. 2025. Provable Benefit of Curriculum in Transformer Tree-Reasoning Post-Training. [arXiv:2511.07372](#) [cs.CL]

- [18] Natasha Butt, Ariel Kwiatkowski, Ismail Labiad, Julia Kempe, and Yann Ollivier. 2025. Soft Tokens, Hard Truths. [arXiv:2509.19170](#) [cs.CL]
- [19] Shuochen Chang, Qingyang Liu, Shaobo Wang, Bingjie Gao, Qianli Ma, Haonan Zhao, Yibo Miao, Yulin Sun, Zelin Peng, Jiangtong Li, and Li Niu. 2026. Why Struggle with Continuous Latents? Interpretable Discrete Latent Reasoning via Rendered Compression. [arXiv:2606.29712](#) [cs.CL]
- [20] Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. Towards Reasoning Era: A Survey of Long Chain-of-Thought for Reasoning Large Language Models. [arXiv preprint arXiv:2503.09567](#). [arXiv:2503.09567](#) [cs.AI]
- [21] Wenhui Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. TheoremQA: A Theorem-driven Question Answering dataset. [arXiv:2305.12524](#) [cs.CL]
- [22] Xin Chen, Hanxian Huang, Yanjun Gao, Yi Wang, Jishen Zhao, and Ke Ding. 2024. Learning to Maximize Mutual Information for Chain-of-Thought Distillation. [arXiv:2403.03348](#) [cs.CL]
- [23] Xinghao Chen, Anhao Zhao, Heming Xia, Xuan Lu, Hanlin Wang, Yanjun Chen, Wei Zhang, Jian Wang, Wenjie Li, and Xiaoyu Shen. 2025. Reasoning Beyond Language: A Comprehensive Survey on Latent Chain-of-Thought Reasoning. [arXiv:2505.16782](#) [cs.CL]
- [24] Zhikang Chen, Sen Cui, Deheng Ye, Yu Zhang, Yatao Bian, and Tingting Zhu. 2025. Think Consistently, Reason Efficiently: Energy-Based Calibration for Implicit Chain-of-Thought. [arXiv:2511.07124](#) [cs.CL]
- [25] Jeffrey Cheng and Benjamin Van Durme. 2024. Compressed Chain of Thought: Efficient Reasoning Through Dense Representations. [arXiv:2412.13171](#) [cs.CL]
- [26] Yunlong Chu, Minglai Shao, Yuhang Liu, Bing Hao, Yumeng Lin, Jialu Wang, and Ruijie Wang. 2026. SPOT: Span-level Pause-of-Thought for Efficient and Interpretable Latent Reasoning in Large Language Models. [arXiv:2603.06222](#) [cs.CL]
- [27] Stephen Chung, Wenyu Du, and Jie Fu. 2025. Thinker: Learning to Think Fast and Slow. [arXiv:2505.21097](#) [cs.CL]
- [28] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. [arXiv:1803.05457](#) [cs.CL]
- [29] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. [arXiv:2110.14168](#) [cs.CL]
- [30] Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Naveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, et al. 2025. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. [arXiv preprint arXiv:2507.06261](#). [arXiv:2507.06261](#) [cs.CL]
- [31] Aditya Cowsik, Tianyu He, and Andrey Gromov. 2025. Towards Distributed Neural Architectures. [arXiv:2506.22389](#) [cs.LG]
- [32] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. [arXiv preprint arXiv:2501.12948](#). [arXiv:2501.12948](#) [cs.CL]
- [33] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, et al. 2024. DeepSeek-V3 Technical Report. [arXiv preprint arXiv:2412.19437](#). [arXiv:2412.19437](#) [cs.CL]
- [34] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2018. Universal Transformers. [arXiv preprint arXiv:1807.03819](#). [arXiv:1807.03819](#) [cs.CL]
- [35] Huilin Deng, Hongchen Luo, Yue Zhu, Long Li, Zhuoyue Chen, Xinghao Zhao, Ming Li, Jihai Zhang, Mengchang Wang, Yang Cao, and Yu Kang. 2026. IIB-LPO: Latent Policy Optimization via Iterative Information Bottleneck. [arXiv:2601.05870](#) [cs.CL]
- [36] Yuntian Deng, Yejin Choi, and Stuart Shieber. 2024. From Explicit CoT to Implicit CoT: Learning to Internalize CoT Step by Step. [arXiv:2405.14838](#) [cs.CL]
- [37] Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. 2023. Implicit Chain of Thought Reasoning via Knowledge Distillation. [arXiv:2311.01460](#) [cs.CL]
- [38] Qingxiu Dong, Li Dong, Yao Tang, Tianzhu Ye, Yutao Sun, Zhifang Sui, and Furu Wei. 2025. Reinforcement Pre-Training. [arXiv preprint arXiv:2506.08007](#). [arXiv:2506.08007](#) [cs.CL]
- [39] Nikita Dragunov, Temurbek Rahmatullaev, Elizaveta Goncharova, Nikita Kurdiukov, Aysel Mirzoeva, Anna Borisiuk, Andrey Kuznetsov, and Anton Razzhigaev. 2025. SONAR-LLM: Autoregressive Transformer that Thinks in Sentence Embeddings and Speaks in Tokens. [arXiv:2508.05305](#) [cs.CL]
- [40] Y. Du, C. Guo, W. Wang, and G. Tang. 2025. Cognitive Decision Routing in Large Language Models: When to Think Fast, When to Think Slow. [arXiv:2508.16636](#) [cs.CL]
- [41] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. [arXiv:1903.00161](#) [cs.CL]
- [42] Björn Engdahl, Adrian Kosowski, Jan Chorowski, Zuzanna Stamirowska, Przemysław Uznański, Junlin Jiang, Rohan Phadke, Remigiusz Kinas, and Richard Zhong. 2026. BDH-CQ: In-Context Learning with Recurrent Latent Reasoning. [arXiv:2608.09888](#) [cs.LG]
- [43] Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. 2024. Looped Transformers for Length Generalization. [arXiv:2409.15647](#) [cs.LG]
- [44] Ying Fan, Anej Svete, and Kangwook Lee. 2026. Bridging the Gap Between Latent and Explicit Reasoning with Looped Transformers. [arXiv:2606.31779](#) [cs.LG]
- [45] Mujtaba Farhan and Maheep Chaudhary. 2026. Why Limit the Residual Stream to Layers and Not Tokens? Persistent Memory for Continuous Latent Reasoning. [arXiv:2606.07720](#) [cs.CL]

- [46] Jacob Fein-Ashley and Paria Rashidinejad. 2026. Solve the Loop: Attractor Models for Language and Reasoning. arXiv:2605.12466 [cs.LG]
- [47] Shengyu Feng, Yun He, Shuang Ma, Beibin Li, Yuanhao Xiong, Songlin Li, Karishma Mandyam, Julian Katz-Samuels, Shengjie Bi, Licheng Yu, Hejia Zhang, Karthik Abinav Sankararaman, Han Fang, Yiming Yang, and Manaal Faruqui. 2025. Bradley-Terry Policy Optimization for Generative Preference Modeling. arXiv:2510.15242 [cs.CL]
- [48] Markus Frey, Behzad Shomali, Ali Hamza Bashir, David Berghaus, Joachim Koehler, and Mehdi Ali. 2026. Adaptive Loops and Memory in Transformers: Think Harder or Know More? arXiv:2603.08391 [cs.LG]
- [49] Hengyu Fu, Tianyu Guo, Zixuan Wang, Hanlin Zhu, Jason D. Lee, Jiantao Jiao, Stuart Russell, and Song Mei. 2026. DiscoLoop: Looping Discrete Embeddings and Continuous Hidden States for Multi-hop Reasoning. arXiv:2607.00341 [cs.LG]
- [50] Rao Fu, Zixuan Yang, Jiankun Zhang, Jing Ma, Hechang Chen, Yu Li, and Yi Chang. 2026. Simply Stabilizing the Loop via Fully Looped Transformer. arXiv:2605.18797 [cs.LG]
- [51] Alexandre Galashov, Matt Jones, Rosemary Ke, Yuan Cao, Vaishnavh Nagarajan, and Michael C. Mozer. 2025. Catch Your Breath: Adaptive Computation for Self-Paced Sequence Production. arXiv:2510.13879 [cs.CL]
- [52] Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. 2024. Stream of Search (SoS): Learning to Search in Language. arXiv:2404.03683 [cs.LG]
- [53] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. arXiv preprint arXiv:2101.00027. arXiv:2101.00027 [cs.CL]
- [54] Zitian Gao, Yilong Chen, Yihao Xiao, Xinyu Yang, Ran Tao, Joey Zhou, and Bryan Dai. 2026. Loop the Loopies! arXiv:2607.16051 [cs.LG]
- [55] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. arXiv preprint arXiv:2502.05171. arXiv:2502.05171 [cs.LG]
- [56] Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies. arXiv:2101.02235 [cs.CL]
- [57] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. Better & Faster Large Language Models via Multi-token Prediction. arXiv:2404.19737 [cs.CL]
- [58] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. 2023. Think before you speak: Training Language Models With Pause Tokens. arXiv preprint arXiv:2310.02226. arXiv:2310.02226 [cs.CL]
- [59] Xinyu Guan, Li Lina Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rStar-Math: Small LLMs Can Master Math Reasoning with Self-Evolved Deep Thinking. arXiv:2501.04519 [cs.CL]
- [60] Shibo Hao, Sainbayar Sukhbaatar, Dijia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training Large Language Models to Reason in a Continuous Latent Space. arXiv preprint arXiv:2412.06769. arXiv:2412.06769 [cs.CL]
- [61] Ali Hatamizadeh, Syeda Nahida Akter, Shrimai Prabhumoye, Jan Kautz, Mostafa Patwary, Mohammad Shoeibi, Bryan Catanzaro, and Yejin Choi. 2025. RLP: Reinforcement as a Pretraining Objective. arXiv preprint arXiv:2510.01265. arXiv:2510.01265 [cs.LG]
- [62] Ali Hatamizadeh, Shrimai Prabhumoye, Igor Gitman, Ximing Lu, Seungju Han, Wei Ping, Yejin Choi, and Jan Kautz. 2026. iGRPO: Self-Feedback-Driven LLM Reasoning. arXiv:2602.09000 [cs.CL]
- [63] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. OlympiadBench: A Challenging Benchmark for Promoting AGI with Olympiad-Level Bilingual Multimodal Scientific Problems. arXiv:2402.14008 [cs.CL]
- [64] Hongyang He, Jiuming Liu, and Victor Sanchez. 2026. Revisiting Chain-of-Thought Reasoning under Limited Supervision: Semi-supervised Chain-of-Thought Learning. arXiv:2607.01511 [cs.CL]
- [65] Amr Hegazy, Amr Alanwar, and Mostafa Elhoushi. 2026. Gated Recurrent Transformers: Expressive Depth through Recurrent Modulation. arXiv:2608.15062 [cs.LG]
- [66] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring Massive Multitask Language Understanding. arXiv:2009.03300 [cs.CL]
- [67] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring Mathematical Problem Solving With the MATH Dataset. arXiv:2103.03874 [cs.CL]
- [68] David Herel and Tomas Mikolov. 2024. Thinking Tokens for Language Modeling. arXiv preprint arXiv:2405.08644. arXiv:2405.08644 [cs.CL]
- [69] Chia-Hsuan Hsu and Jui-Ming Yao. 2026. Learning to Refine Hidden States for Reliable LLM Reasoning. arXiv:2606.17524 [cs.CL]
- [70] Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. 2025. Open-Reasoner-Zero: An Open Source Approach to Scaling Up Reinforcement Learning on the Base Model. arXiv preprint arXiv:2503.24290. arXiv:2503.24290 [cs.LG]
- [71] Wei Huang, Yizhe Xiong, Xin Ye, Zhijie Deng, Hui Chen, Zijia Lin, and Guiguang Ding. 2025. Fast Quiet-STaR: Thinking Without Thought Tokens. arXiv:2505.17746 [cs.CL]
- [72] Jaber Jaber and Osama Jaber. 2026. Ouroboros: Dynamic Weight Generation for Recursive Transformers via Input-Conditioned LoRA Modulation. arXiv:2604.02051 [cs.CL]
- [73] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. arXiv:2403.07974 [cs.CL]

- [74] Yixin Ji, Juntao Li, Yang Xiang, Hai Ye, Kaixin Wu, Kai Yao, Jia Xu, Linjian Mo, and Min Zhang. 2025. A Survey of Test-Time Compute: From Intuitive Inference to Deliberate Reasoning. arXiv preprint arXiv:2501.02497. arXiv:2501.02497 [cs.AI]
- [75] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770 [cs.CL]
- [76] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2020. What Disease does this Patient Have? A Large-scale Open Domain Question Answering Dataset from Medical Exams. arXiv:2009.13081 [cs.CL]
- [77] Mingyu Jin, Weidi Luo, Sitao Cheng, Xinyi Wang, Wenyue Hua, Ruixiang Tang, William Yang Wang, and Yongfeng Zhang. 2024. Disentangling Memory and Reasoning Ability in Large Language Models. arXiv:2411.13504 [cs.CL]
- [78] Wooll Jung. 2026. Dropout-GRPO: Variational Stochasticity for Continuous Latent Reasoning. arXiv:2606.10184 [cs.CL]
- [79] Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Yi-An Ma, and Lianhui Qin. 2026. LaDi-RL: Latent Diffusion Reasoning Prevents Entropy Collapse in Reinforcement Learning. arXiv:2602.01705 [cs.CL]
- [80] Liwei Kang, Yee Whye Teh, and Wee Sun Lee. 2026. LinTree: Improving LLM Reasoning with Explicitly Structured Search Histories. arXiv:2605.31492 [cs.CL]
- [81] Shashank Kapadia, Deep Naryan Mishra, Sujal Reddy Alugubelli, Haoan Wang, Saipraveen Vabbilisetty, Rishi Bhatia, and Anupriya Sharma. 2026. LEAP: Layer-wise Exit-Aware Pretraining for Efficient Transformer Inference. arXiv:2605.01058 [cs.CL]
- [82] Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, Peifeng Wang, Silvio Savarese, Caiming Xiong, and Shafiq Joty. 2025. A Survey of Frontiers in LLM Reasoning: Inference Scaling, Learning to Reason, and Agentic Systems. arXiv preprint arXiv:2504.09037. arXiv:2504.09037 [cs.AI]
- [83] Bum Jun Kim, Kohei Hayashi, Shunsuke Kamiya, Masanori Koyama, Yusuke Iwasawa, and Yutaka Matsuo. 2026. Looped Transformers with Source-Centered State Evolution. arXiv:2607.27656 [cs.LG]
- [84] Eunki Kim, Sangryul Kim, and James Thorne. 2025. Learning to Insert [PAUSE] Tokens for Better Reasoning. arXiv:2506.03616 [cs.CL]
- [85] Juno Kim and Taiji Suzuki. 2024. Transformers Provably Solve Parity Efficiently with Chain of Thought. arXiv:2410.08633 [cs.CL]
- [86] Jonas Knupp, Jan Hendrik Metzen, Jeremias Bohn, Georg Groh, and Kristian Kersting. 2026. Depth-Recurrent Attention Mixtures: Giving Latent Reasoning the Attention it Deserves. arXiv:2601.21582 [cs.LG]
- [87] Yeskendir Koishekenov, Aldo Lipani, and Nicola Cancedda. 2025. Encode, Think, Decode: Scaling test-time reasoning with recursive latent thoughts. arXiv:2510.07358 [cs.CL]
- [88] Deqian Kong, Minglu Zhao, Dehong Xu, Bo Pang, Shu Wang, Edouardo Honig, Zhangzhang Si, Chuan Li, Jianwen Xie, Sirui Xie, and Ying Nian Wu. 2025. Latent Thought Models with Variational Bayes Inference-Time Computation. arXiv:2502.01567 [cs.CL]
- [89] Nikita Koriagin, Yaroslav Aksenov, George Bredis, Gleb Gerasimov, Nikita Balagansky, and Daniil Gavrilov. 2026. A Model with No Head and Many Thoughts. arXiv:2608.31069 [cs.CL]
- [90] Komal Kumar, Tajamul Ashraf, Omark Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip H.S. Torr, Fahad Shabbaz Khan, and Salman Khan. 2025. LLM Post-Training: A Deep Dive into Reasoning Large Language Models. arXiv:2502.21321 [cs.CL]
- [91] Shashi Kumar, Yacouba Kaloga, Petr Motlicek, Ina Kodrasi, and Andrea Cavallaro. 2026. Geometric Latent Reasoning Induces Shorter Generations in LLMs. arXiv:2606.02248 [cs.CL]
- [92] Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. 2024. Step-DPO: Step-wise Preference Optimization for Long-chain Reasoning of LLMs. arXiv:2406.18629 [cs.CL]
- [93] Filipe Laitenberger, Dawid Kopiczko, Cees G.M. Snoek, and Yuki M. Asano. 2025. What Layers When: Learning to Skip Compute in LLMs with Residual Gates. arXiv:2510.13876 [cs.CL]
- [94] Jack Lanchantin, Shubham Toshniwal, Jason Weston, Arthur Szlam, and Sainbayar Sukhbaatar. 2023. Learning to Reason and Memorize with Self-Notes. arXiv:2305.00833 [cs.CL]
- [95] Tim Lawson and Laurence Aitchison. 2025. Learning to Skip the Middle Layers of Transformers. arXiv:2506.21103 [cs.LG]
- [96] Ryan Lee, Jacob Biloki, Edward J. Hu, and Jonathan May. 2026. Sparse Layers are Critical to Scaling Looped Language Models. arXiv:2605.09165 [cs.LG]
- [97] Yookyung Lee, Yumeng Shen, Jinhyeong Park, Hyein Yang, and Jun-Hyung Park. 2026. CIRF: Tokenizing Chain-of-Thoughts into Reusable Functional Units for Efficient Latent Reasoning in Large Language Models. arXiv:2605.28292 [cs.CL]
- [98] Lucas Lehnert, Sainbayar Sukhbaatar, Dijia Su, Qingqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. 2024. Beyond A*: Better Planning with Transformers via Search Dynamics Bootstrapping. arXiv:2402.14083 [cs.LG]
- [99] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving Quantitative Reasoning Problems with Language Models. arXiv:2206.14858 [cs.CL]
- [100] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, et al. 2024. DataComp-LM: In search of the next generation of training sets for language models. arXiv preprint arXiv:2406.11794. arXiv:2406.11794 [cs.LG]
- [101] Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. 2023. Symbolic Chain-of-Thought Distillation: Small Models Can Also "Think" Step-by-Step. arXiv preprint arXiv:2306.14050. arXiv:2306.14050 [cs.CL]

- [102] Siheng Li, Kejiao Li, Zenan Xu, Guanhua Huang, Evander Yang, Kun Li, Haoyuan Wu, Jiajia Wu, Zihao Zheng, Chenchen Zhang, Kun Shi, Kyriell Deng, Qi Yi, Ruibin Xiong, Tingqiang Xu, Yuhao Jiang, Jianfeng Yan, Yuyuan Zeng, Guanghui Xu, Jinbao Xue, Zhijiang Xu, Zheng Fang, Shuai Li, Qibin Liu, Xiaoxue Li, Zhuoyu Li, Yangyu Tao, Fei Gao, Cheng Jiang, Bo Chao Wang, Kai Liu, Jianchen Zhu, Wai Lam, Wayyt Wang, Bo Zhou, and Di Wang. 2025. Reinforcement Learning on Pre-Training Data. [arXiv:2509.19249](#) [cs.CL]
- [103] Yu Li, Shu Hong, and Tian Lan. 2026. Localizing Credit at the Divergence: Path-Conditioned Self-Distillation for LLM Reasoning. [arXiv:2606.15576](#) [cs.CL]
- [104] Ziyue Li, Yang Li, and Tianyi Zhou. 2025. Skip a Layer or Loop it? Test-Time Depth Adaptation of Pretrained LLMs. [arXiv:2507.07996](#) [cs.CL]
- [105] Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. 2024. Chain of Thought Empowers Transformers to Solve Inherently Serial Problems. [arXiv preprint arXiv:2402.12875](#). [arXiv:2402.12875](#) [cs.LG]
- [106] Zihang Li, Yuhang Wang, Yikun Zong, Wenhan Yu, Xiaokun Yuan, Runhan Jiang, Zirui Liu, Tong Yang, and Arthur Jiang. 2026. EntroCoT: Enhancing Chain-of-Thought via Adaptive Entropy-Guided Segmentation. [arXiv:2601.03769](#) [cs.CL]
- [107] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's Verify Step by Step. [arXiv:2305.20050](#) [cs.CL]
- [108] Bill Yuchen Lin, Ronan Le Bras, Kyle Richardson, Ashish Sabharwal, Radha Poovendran, Peter Clark, and Yejin Choi. 2025. ZebraLogic: On the Scaling Limits of LLMs for Logical Reasoning. [arXiv:2502.01100](#) [cs.CL]
- [109] Hanyu Lin, Min Cai, Jiawei Wen, and Haodi Zhang. 2026. Tyler: Typed Latent Reasoning for Language Models – When to Think, What to Compute, and How Much to Allocate. [arXiv:2606.16360](#) [cs.CL]
- [110] Aiwei Liu, Cheng Shi, Chuhan Wu, Ci Lei, Di Lu, Donald He, Fan Zhang, Fanhao Kong, Feifei Zhang, Guan Wang, Haicheng Wang, Haoyu Liu, Houjin Yu, Jiachen Ding, Jiayi Feng, Jie Zhou, Jijun Chi, Jindi Shi, Jing Lei, Junjie Zhang, Laiyi Li, Le Tian, Linhao Zhang, Miao Fan, Sijun Zhang, Wei Jia, Weiwei Shi, Wenhan Li, Wentao Zhao, Wenteng Liang, Xiao Zhou, Xiaojin Zhou, Xihuai Wang, Xinyu Gao, Xuanliang Wang, Xuyang Ao, Yang Yu, Yangxiu You, Yinyao Zhao, Yufei Kuang, Yufei Wang, Yuan Liu, Yuan Liu, Yuwen Chen, Zhencong Tian, Zhongyin Zhao, Zilin Yu, and Zitao Wang. 2026. Hidden Decoding at Scale: Latent Computation Scaling for Large Language Models. [arXiv:2607.08186](#) [cs.CL]
- [111] Jiayu Liu, Zhenya Huang, Xuan Yang, Tianyun Ji, Anya Sims, Hao Xu, Enhong Chen, Yee Whye Teh, and Ning Miao. 2025. Deep Thinking by Markov Chain of Continuous Thoughts. [arXiv:2509.25020](#) [cs.CL]
- [112] Kelian Liu, Dingkan Yang, Ziyun Qian, Weijie Yin, Yuchi Wang, Hongsheng Li, Jun Liu, Peng Zhai, Yang Liu, and Lihua Zhang. 2025. Reinforcement Learning Meets Large Language Models: A Survey of Advancements and Applications Across the LLM Lifecycle. [arXiv preprint arXiv:2509.16679](#). [arXiv:2509.16679](#) [cs.CL]
- [113] Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. 2025. ProRL: Prolonged Reinforcement Learning Expands Reasoning Boundaries in Large Language Models. [arXiv:2505.24864](#) [cs.CL]
- [114] Weihao Liu, Dehai Min, and Lu Cheng. 2026. Latent Thoughts Tuning: Bridging Context and Reasoning with Fused Information in Latent Tokens. [arXiv:2602.10229](#) [cs.CL]
- [115] Xin Liu, Runsong Zhao, Xinyu Liu, Junhao Ruan, Pengcheng Huang, Shichao Dong, Chunyang Xiao, Chenglong Wang, Changliang Li, Jingbo Zhu, and Tong Xiao. 2026. HybridThinker: Efficient Chain-of-Thought Reasoning via Compressed Memory and Transient Thought Steps. [arXiv:2606.03768](#) [cs.CL]
- [116] Yulian Liu, Yunchong Song, Yixuan Wang, Kewen Ge, Alex Lamb, Qipeng Guo, Kai Chen, Bowen Zhou, and Zhouhan Lin. 2026. Next Concept Prediction in Discrete Latent Space Leads to Stronger Language Models. [arXiv:2602.08984](#) [cs.CL]
- [117] Joe Logan. 2026. Per-Token Fixed-Point Convergence in Depth-Recurrent Transformers. [arXiv:2607.14427](#) [cs.LG]
- [118] Charles London and Varun Kanade. 2025. Pause Tokens Strictly Increase the Expressivity of Constant-Depth Transformers. [arXiv preprint arXiv:2505.21024](#). [arXiv:2505.21024](#) [cs.LG]
- [119] Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. 2022. Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering. [arXiv:2209.09513](#) [cs.CL]
- [120] Wenquan Lu, Yuechuan Yang, Kyle Lee, Yanshu Li, and Enqi Liu. 2025. Latent Chain-of-Thought? Decoding the Depth-Recurrent Transformer. [arXiv:2507.02199](#) [cs.LG]
- [121] Zimu Lu, Aojun Zhou, Ke Wang, Houxing Ren, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. 2024. MathCoder2: Better Math Reasoning from Continued Pretraining on Model-translated Mathematical Code. [arXiv:2410.08196](#) [cs.CL]
- [122] Qin-Wen Luo, Sheng Ren, Xiang Chen, Rui Liu, Jun Fang, Naiqiang Tan, and Sheng-Jun Huang. 2026. Compress the Easy, Explore the Hard: Difficulty-Aware Entropy Regularization for Efficient LLM Reasoning. [arXiv:2602.22642](#) [cs.CL]
- [123] Xiaocheng Luo, Kang Wang, Zaifu Zhan, Yuechi Zhou, and Xiangyu Duan. 2026. RuPLaR: Efficient Latent Compression of LLM Reasoning Chains with Rule-Based Priors From Multi-Step to One-Step. [arXiv:2605.09346](#) [cs.CL]
- [124] Divyat Mahajan, Sachin Goyal, Badr Youbi Idrissi, Mohammad Pezeshki, Ioannis Mitliagkas, David Lopez-Paz, and Kartik Ahuja. 2025. Beyond Multi-Token Prediction: Pretraining LLMs with Future Summaries. [arXiv:2510.14751](#) [cs.LG]
- [125] Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. 2025. Teaching Pretrained Language Models to Think Deeper with Retrofitted Recurrence. [arXiv:2511.07384](#) [cs.CL]
- [126] William Merrill and Ashish Sabharwal. 2023. The Expressive Power of Transformers with Chain of Thought. [arXiv preprint arXiv:2310.07923](#). [arXiv:2310.07923](#) [cs.LG]

- [127] Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2021. A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers. [arXiv:2106.15772](#) [cs.CL]
- [128] Ahmed Abdelmunim Abdalla Mohammed. 2026. Adaptive Computation Depth via Learned Token Routing in Transformers. [arXiv:2605.05222](#) [cs.LG]
- [129] Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T. Konstantin Rusch, and Antonio Orvieto. 2026. Fixed-Point Reasoners: Stable and Adaptive Deep Looped Transformers. [arXiv:2606.18206](#) [cs.LG]
- [130] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. [arXiv preprint arXiv:2501.19393](#). [arXiv:2501.19393](#) [cs.CL]
- [131] Alex Ning, Yen-Ling Kuo, and Gabe Gomes. 2025. Learning When to Stop: Adaptive Latent Reasoning via Reinforcement Learning. [arXiv:2511.21581](#) [cs.CL]
- [132] Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, et al. 2025. Olmo 3. [arXiv preprint arXiv:2512.13961](#). [arXiv:2512.13961](#) [cs.CL]
- [133] Team OLmo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, et al. 2024. 2 OLmo 2 Furious. [arXiv preprint arXiv:2501.00656](#). [arXiv:2501.00656](#) [cs.CL]
- [134] OpenAI, ., Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. OpenAI o1 System Card. [arXiv preprint arXiv:2412.16720](#). [arXiv:2412.16720](#) [cs.AI]
- [135] Abhishek Panwar, Maheep Singh, and Saksham Bansal. 2026. Think Deep, Speak Once: Relit, A Recursive Latent Implicit Transformer Framework. [arXiv:2608.08113](#) [cs.CL]
- [136] Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, and Shuiwang Ji. 2025. Curriculum Reinforcement Learning from Easy to Hard Tasks Improves LLM Reasoning. [arXiv:2506.06632](#) [cs.CL]
- [137] Taekhyun Park, Yongjae Lee, Dohee Kim, and Hyerim Bae. 2026. LoopUS: Recasting Pretrained LLMs into Looped Latent Refinement Models. [arXiv:2605.11011](#) [cs.CL]
- [138] Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2023. OpenWebMath: An Open Dataset of High-Quality Mathematical Web Text. [arXiv preprint arXiv:2310.06786](#). [arXiv:2310.06786](#) [cs.AI]
- [139] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP Models really able to Solve Simple Math Word Problems? [arXiv:2103.07191](#) [cs.CL]
- [140] Elizabeth Pavlova, Mariia Koroliuk, Karthik Viswanathan, Cameron Tice, Edward James Young, and Puria Radmard. 2026. A transformer architecture alteration to incentivise externalised reasoning. [arXiv:2603.21376](#) [cs.CL]
- [141] Guilherme Penedo, Hynek Kydliček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. [arXiv preprint arXiv:2406.17557](#). [arXiv:2406.17557](#) [cs.CL]
- [142] Jacob Pfau, William Merrill, and Samuel R. Bowman. 2024. Let's Think Dot by Dot: Hidden Computation in Transformer Language Models. [arXiv preprint arXiv:2404.15758](#). [arXiv:2404.15758](#) [cs.CL]
- [143] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. 2025. Humanity's Last Exam. [arXiv:2501.14249](#) [cs.CL]
- [144] Shengmin Piao and Sanghyun Park. 2025. SpiralThinker: Latent Reasoning through an Iterative Process with Text-Latent Interleaving. [arXiv:2511.08983](#) [cs.CL]
- [145] Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y. Fu. 2026. Parcae: Scaling Laws For Stable Looped Language Models. [arXiv:2604.12946](#) [cs.LG]
- [146] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. [arXiv:2307.16789](#) [cs.AI]
- [147] Xiaoye Qu, Yafu Li, Zhao-Chen Su, Weigao Sun, Jianhao Yan, Dongrui Liu, Ganqu Cui, Daizong Liu, Shuxian Liang, Junxian He, Peng Li, Wei Wei, et al. 2025. A Survey of Efficient Reasoning for Large Reasoning Models: Language, Multimodality, and Beyond. [arXiv preprint arXiv:2503.21614](#). [arXiv:2503.21614](#) [cs.CL]
- [148] Xingwei Qu, Shaowen Wang, Zihao Huang, Kai Hua, Fan Yin, Rui-Jie Zhu, Jundong Zhou, Qiyang Min, Zihao Wang, Yizhi Li, Tianyu Zhang, He Xing, Zheng Zhang, Yuxuan Song, Tianyu Zheng, Zhiyuan Zeng, Chenghua Lin, Ge Zhang, and Wenhao Huang. 2025. Dynamic Large Concept Models: Latent Reasoning in an Adaptive Semantic Space. [arXiv:2512.24617](#) [cs.CL]
- [149] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. [arXiv preprint arXiv:1910.10683](#). [arXiv:1910.10683](#) [cs.LG]
- [150] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-Depths: Dynamically allocating compute in transformer-based language models. [arXiv:2404.02258](#) [cs.LG]
- [151] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. [arXiv:2311.12022](#) [cs.CL]
- [152] Yangjun Ruan, Neil Band, Chris J. Maddison, and Tatsunori Hashimoto. 2025. Reasoning to Learn from Latent Thoughts. [arXiv:2503.18866](#) [cs.CL]
- [153] Abulhair Saparov and He He. 2022. Language Models Are Greedy Reasoners: A Systematic Formal Analysis of Chain-of-Thought. [arXiv:2210.01240](#) [cs.CL]

- [154] Grigory Sapunov. 2026. Universal Transformers Need Memory: Depth-State Trade-offs in Adaptive Recursive Reasoning. arXiv:2604.21999 [cs.CL]
- [155] Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761 [cs.CL]
- [156] Hongyu Shan, Mingyang Song, Chang Dai, Di Liang, and Han Chen. 2025. R-Capsule: Compressing High-Level Plans for Efficient Large Language Model Reasoning. arXiv:2509.22131 [cs.CL]
- [157] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv preprint arXiv:2402.03300. arXiv:2402.03300 [cs.CL]
- [158] Mark Shapiro. 2026. Retrofitting Recurrent Depth into a Pretrained Language Model: Installation, Extrapolation, Transfer, and Retention at Two Parameter Budgets. arXiv:2608.11233 [cs.CL]
- [159] Jingyan Shen, Ang Li, Salman Rahman, Yifan Sun, Micah Goldblum, Matus Telgarsky, and Pavel Izmailov. 2026. Understanding Reasoning from Pretraining to Post-Training. arXiv preprint arXiv:2607.16097. arXiv:2607.16097 [cs.LG]
- [160] Zhiqiang Shen, Tianhua Tao, Liqun Ma, Willie Neiswanger, Zhengzhong Liu, Hongyi Wang, Bowen Tan, Joel Hestness, Natalia Vassilieva, Daria Soboleva, and Eric Xing. 2023. SlimPajama-DC: Understanding Data Combinations for LLM Training. arXiv preprint arXiv:2309.10818. arXiv:2309.10818 [cs.CL]
- [161] Behzad Shomali, Markus Frey, David Berghaus, Joachim Koehler, and Mehdi Ali. 2026. LoopMTP: A looped transformer guided by latent multi-token prediction. arXiv:2608.03624 [cs.LG]
- [162] Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. 2023. Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models. arXiv:2312.06585 [cs.CL]
- [163] Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, et al. 2024. Dolma: an Open Corpus of Three Trillion Tokens for Language Model Pretraining Research. arXiv preprint arXiv:2402.00159. arXiv:2402.00159 [cs.CL]
- [164] Ryan Solgi, Jiayi Tian, and Zheng Zhang. 2026. LoRi: Low-Rank Distillation for Implicit Reasoning. arXiv:2606.05315 [cs.CL]
- [165] Shixiang Song, He Li, Zitong Wang, Boyi Zeng, Feichen Song, Yixuan Wang, Zhiqin John Xu, Ziwei He, and Zhouhan Lin. 2026. AdaPonderLM: Gated Pondering Language Models with Token-Wise Adaptive Depth. arXiv:2603.01914 [cs.CL]
- [166] Zayne Sprague, Xi Ye, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. 2023. MuSR: Testing the Limits of Chain-of-thought with Multistep Soft Reasoning. arXiv:2310.16049 [cs.CL]
- [167] Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. 2024. Nemotron-CC: Transforming Common Crawl into a Refined Long-Horizon Pretraining Dataset. arXiv preprint arXiv:2412.02595. arXiv:2412.02595 [cs.CL]
- [168] Dijia Su, Sainbayar Sukhbaatar, Michael Rabbat, Yuandong Tian, and Qingqing Zheng. 2024. Dualformer: Controllable Fast and Slow Thinking by Learning with Randomized Reasoning Traces. arXiv:2410.09918 [cs.CL]
- [169] Dijia Su, Hanlin Zhu, Yingchen Xu, Jiantao Jiao, Yuandong Tian, and Qingqing Zheng. 2025. Token Assorted: Mixing Latent and Text Tokens for Improved Language Model Reasoning. arXiv:2502.03275 [cs.CL]
- [170] Ayhan Suleymanzade, Halil Alperen Gozeten, Michael Bronstein, İsmail İlkan Ceylan, and Jinwoo Kim. 2026. MUX: Continuous Reasoning via Multiplexed Tokens. arXiv:2607.18264 [cs.CL]
- [171] Yuchang Sun, Yanxi Chen, Yaliang Li, and Bolin Ding. 2025. Enhancing Latent Computation in Transformers with Latent Tokens. arXiv:2505.12629 [cs.CL]
- [172] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2022. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. arXiv:2210.09261 [cs.CL]
- [173] Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason Weston, and Xian Li. 2025. LLM Pretraining with Continuous Concepts. arXiv preprint arXiv:2502.08524. arXiv:2502.08524 [cs.LG]
- [174] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. arXiv:1811.00937 [cs.CL]
- [175] Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. 2023. Sparse Universal Transformer. arXiv:2310.07096 [cs.LG]
- [176] Guo Tang, Shixin Jiang, Heng Chang, Nuo Chen, Yuhua Li, Huiming Fan, Jia Li, Ming Liu, and Bing Qin. 2026. LoopRPT: Reinforcement Pre-Training for Looped Language Models. arXiv:2603.19714 [cs.CL]
- [177] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chunong Tang, Congcong Wang, et al. 2025. Kimi k1.5: Scaling Reinforcement Learning with LLMs. arXiv preprint arXiv:2501.12599. arXiv:2501.12599 [cs.AI]
- [178] Guancheng Tu, Xiangjun Fu, Suhao Yu, Yao Tang, Haoqiang Kang, Lianhui Qin, Yizhe Zhang, and Jiatao Gu. 2026. Latent Reasoning with Normalizing Flows. arXiv:2606.06447 [cs.CL]

- [179] Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. [arXiv:2206.10498](#) [cs.CL]
- [180] Dayu Wang, Jiaye Yang, Weikang Li, Jiahui Liang, and Yang Li. 2026. Beyond Alignment: Expanding Reasoning Capacity via Manifold-Reshaping Policy Optimization. [arXiv:2602.02545](#) [cs.CL]
- [181] Fanmeng Wang, Haotian Liu, Guojiang Zhao, Hongteng Xu, and Zhifeng Gao. 2026. ReGuLaR: Variational Latent Reasoning Guided by Rendered Chain-of-Thought. [arXiv:2601.23184](#) [cs.CL]
- [182] Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. 2025. Hierarchical Reasoning Model. [arXiv:2506.21734](#) [cs.CL]
- [183] Jiaqi Wang, Binqian Ji, Haibo Luo, Yiyang Qi, Ruiting Li, Huiyan Wang, Yuantao Han, Cangyi Yang, jiaxu Zhang, and Feiliang Ren. 2025. LTA-thinker: Latent Thought-Augmented Training Framework for Large Language Models on Complex Reasoning. [arXiv:2509.12875](#) [cs.CL]
- [184] Jiahao Wang, Bingyu Liang, Chenhao Hu, Longhui Zhang, Xuebo Liu, Min zhang, Jing Li, and Xuelong Li. 2026. SuCo: Sufficiency-guided Continuous Adaptive Reasoning. [arXiv:2606.17687](#) [cs.CL]
- [185] Kang Wang, Xiangyu Duan, and Tianyi Du. 2025. Improving Latent Reasoning in LLMs via Soft Concept Mixing. [arXiv:2511.16885](#) [cs.CL]
- [186] Liang Wang, Nan Yang, Shaohan Huang, Li Dong, and Furu Wei. 2025. Thinking Augmented Pre-training. [arXiv:2509.20186](#) [cs.CL]
- [187] Peiyi Wang, Lei Li, Zhihong Shao, R.X. Xu, Damai Dai, Yifei Li, Deli Chen, Y.Wu, and Zhifang Sui. 2023. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. [arXiv:2312.08935](#) [cs.CL]
- [188] Xinyi Wang, Lucas Caccia, Oleksiy Ostapenko, Xingdi Yuan, William Yang Wang, and Alessandro Sordani. 2023. Guiding Language Model Reasoning with Planning Tokens. [arXiv:2310.05707](#) [cs.CL]
- [189] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. 2024. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. [arXiv:2407.16741](#) [cs.SE]
- [190] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhramil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhui Chen. 2024. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark. [arXiv:2406.01574](#) [cs.CL]
- [191] Yubo Wang, Ping Nie, Kai Zou, Lijun Wu, and Wenhui Chen. 2025. Unleashing the Reasoning Potential of Pre-trained LLMs by Critique Fine-Tuning on One Problem. [arXiv:2506.03295](#) [cs.CL]
- [192] Zengzhi Wang, Xuefeng Li, Rui Xia, and Pengfei Liu. 2023. MathPile: A Billion-Token-Scale Pretraining Corpus for Math. [arXiv preprint arXiv:2312.17120](#). [arXiv:2312.17120](#) [cs.CL]
- [193] Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. 2025. OctoThinker: Mid-training Incentivizes Reinforcement Learning Scaling. [arXiv:2506.20512](#) [cs.LG]
- [194] Maurice Weber, Daniel Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, et al. 2024. RedPajama: an Open Dataset for Training Large Language Models. [arXiv preprint arXiv:2411.12372](#). [arXiv:2411.12372](#) [cs.CL]
- [195] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. [arXiv preprint arXiv:2201.11903](#). [arXiv:2201.11903](#) [cs.CL]
- [196] Xilin Wei, Xiaoran Liu, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Jiaqi Wang, Xipeng Qiu, and Dahua Lin. 2025. SIM-CoT: Supervised Implicit Chain-of-Thought. [arXiv preprint arXiv:2509.20317](#). [arXiv:2509.20317](#) [cs.CL]
- [197] Frederico Wieser, Martin Benfeghou, Haitham Bou Ammar, Jun Wang, and Zafeirios Fountas. 2025. Subjective Depth and Timescale Transformers: Learning Where and When to Compute. [arXiv:2511.21408](#) [cs.LG]
- [198] Jonathan Williams and Esin Tureci. 2026. Prioritize the Process, Not Just the Outcome: Rewarding Latent Thought Trajectories Improves Reasoning in Looped Language Models. [arXiv:2602.10520](#) [cs.CL]
- [199] Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. 2025. The Invisible Leash: Why RLVR May or May Not Escape Its Origin. [arXiv:2507.14843](#) [cs.LG]
- [200] Junda Wu, Yuxin Xiong, Xintong Li, Sheldon Yu, Zhengmian Hu, Tong Yu, Rui Wang, Xiang Chen, Jingbo Shang, and Julian McAuley. 2025. CTRLS: Chain-of-Thought Reasoning via Latent State-Transition. [arXiv:2507.08182](#) [cs.CL]
- [201] Violet Xiang, Chase Blagden, Rafael Rafailov, Nathan Lile, Sang Truong, Chelsea Finn, and Nick Haber. 2025. Just Enough Thinking: Efficient Reasoning with Adaptive Length Penalties Reinforcement Learning. [arXiv:2506.05256](#) [cs.CL]
- [202] Zheyang Xiong, Shivam Garg, Max Yu, Vaishnavi Shrivastava, Haoyu Zhao, Anastasios Kyrillidis, and Dimitris Papailiopoulos. 2026. SuperThoughts: Reasoning Tokens in Superposition. [arXiv:2606.13862](#) [cs.CL]
- [203] Fengli Xu, Qianye Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, Chenyang Shao, Yuwei Yan, Qinglong Yang, Yiwen Song, Sijian Ren, Xinyuan Hu, Yu Li, Jie Feng, Chen Gao, and Yong Li. 2025. Towards Large Reasoning Models: A Survey of Reinforced Reasoning with Large Language Models. [arXiv:2501.09686](#) [cs.CL]
- [204] Zhikun Xu, Ming Shen, Jacob Dineen, Zhaonan Li, Xiao Ye, Shijie Lu, Aswin RRV, Chitta Baral, and Ben Zhou. 2024. ToW: Thoughts of Words Improve Reasoning in Large Language Models. [arXiv:2410.16235](#) [cs.CL]

- [205] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, et al. 2025. Qwen3 Technical Report. arXiv preprint arXiv:2505.09388. arXiv:2505.09388 [cs.CL]
- [206] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793 [cs.SE]
- [207] Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. 2023. Looped Transformers are Better at Learning Learning Algorithms. arXiv:2311.12424 [cs.LG]
- [208] Xiao-Wen Yang, Ziyu Han, Xi-Hua Zhang, Wen-Da Wei, Jie-Jing Shao, Lan-Zhe Guo, and Yu-Feng Li. 2026. Stabilizing Recurrent Dynamics for Test-Time Scalable Latent Reasoning in Looped Language Models. arXiv:2605.26733 [cs.LG]
- [209] Zhuoyi Yang, Xu Guo, Tong Zhang, Huijuan Xu, and Boyang Li. 2025. Test-time Scaling of LLMs: A Survey from A Subproblem Structure Perspective. arXiv:2511.14772 [cs.CL]
- [210] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. arXiv:1809.09600 [cs.CL]
- [211] Jiarui Yao, Ruida Wang, Hao Bai, and Tong Zhang. 2026. Future-KL Regularized GRPO: Process-Level Credit Assignment from f -Divergence Regularization. arXiv:2601.10201 [cs.CL]
- [212] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. arXiv:2305.10601 [cs.CL]
- [213] Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. LIMO: Less is More for Reasoning. arXiv preprint arXiv:2502.03387. arXiv:2502.03387 [cs.CL]
- [214] Jingyang Yi, Jiazhen Wang, and Sida Li. 2025. ShorterBetter: Guiding Reasoning Models to Find Optimal Inference Length for Efficient Reasoning. arXiv:2504.21370 [cs.CL]
- [215] Chengting Yu, Xiaobo Shu, Yadao Wang, Yizhen Zhang, Haoyi Wu, Jiaang Li, Rujiao Long, Ziheng Chen, Yuchi Xu, Wenbo Su, and Bo Zheng. 2025. MeSH: Memory-as-State-Highways for Recursive Transformers. arXiv:2510.07739 [cs.LG]
- [216] Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaozhong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. 2025. DAPO: An Open-Source LLM Reinforcement Learning System at Scale. arXiv:2503.14476 [cs.CL]
- [217] Zhenxuan Yu, Takeshi Kojima, Yutaka Matsuo, and Yusuke Iwasawa. 2026. CHASE: Cache-Hole-Adapted Skip Exit for Looped State-Space Language Models. arXiv:2607.10110 [cs.LG]
- [218] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025. Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? arXiv:2504.13837 [cs.AI]
- [219] Zhenrui Yue, Bowen Jin, Huimin Zeng, Honglei Zhuang, Zhen Qin, Jinsung Yoon, Lanyu Shang, Jiawei Han, and Dong Wang. 2025. Hybrid Latent Reasoning via Reinforcement Learning. arXiv:2505.18454 [cs.CL]
- [220] Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. 2024. Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking. arXiv preprint arXiv:2403.09629. arXiv:2403.09629 [cs.CL]
- [221] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. STaR: Bootstrapping Reasoning With Reasoning. arXiv:2203.14465 [cs.LG]
- [222] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? arXiv:1905.07830 [cs.CL]
- [223] Boyi Zeng, Yiqin Hao, He Li, Shixiang Song, Feichen Song, Zitong Wang, Siyuan Huang, Yi Xu, Ziwei He, Xinbing Wang, and Zhouhan Lin. 2026. Pretraining with Token-Level Adaptive Latent Chain-of-Thought. arXiv:2602.08220 [cs.CL]
- [224] Boyi Zeng, He Li, Shixiang Song, Yixuan Wang, Zitong Wang, Ziwei He, Xinbing Wang, and Zhouhan Lin. 2025. PonderLM-2: Pretraining LLM with Latent Thoughts in Continuous Space. arXiv:2509.23184 [cs.CL]
- [225] Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. 2025. PonderLM: Pretraining Language Models to Ponder in Continuous Space. arXiv preprint arXiv:2505.20674. arXiv:2505.20674 [cs.CL]
- [226] Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. 2025. SimpleRL-Zoo: Investigating and Taming Zero Reinforcement Learning for Open Base Models in the Wild. arXiv preprint arXiv:2503.18892. arXiv:2503.18892 [cs.LG]
- [227] Charlie Zhang, Graham Neubig, and Xiang Yue. 2025. On the Interplay of Pre-Training, Mid-Training, and RL on Reasoning Language Models. arXiv preprint arXiv:2512.07783. arXiv:2512.07783 [cs.CL]
- [228] Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search. arXiv:2406.03816 [cs.CL]
- [229] Jinghao Zhang, Naishan Zheng, Ruilin Li, Dongzhou Cheng, Zheming Liang, Feng Zhao, and Jiaqi Wang. 2025. RLFR: Extending Reinforcement Learning for LLMs with Flow Environment. arXiv:2510.10201 [cs.CL]
- [230] Jixiao Zhang and Chunsheng Zuo. 2025. GRPO-LEAD: A Difficulty-Aware Reinforcement Learning Approach for Concise Mathematical Reasoning in Language Models. arXiv:2504.09696 [cs.CL]
- [231] Kongcheng Zhang, Qi Yao, Shunyu Liu, Yingjie Wang, Baisheng Lai, Jieping Ye, Mingli Song, and Dacheng Tao. 2025. Consistent Paths Lead to Truth: Self-Rewarding Reinforcement Learning for LLM Reasoning. arXiv:2506.08745 [cs.CL]

- [232] Liting Zhang, Shiwan Zhao, Xuyang Zhao, Zichen Xu, Jianye Wang, and Qicheng Li. 2026. TARPO: Token-Wise Latent-Explicit Reasoning via Action-Routing Policy Optimization. [arXiv:2606.05859](#) [cs.CL]
- [233] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. TinyLlama: An Open-Source Small Language Model. [arXiv preprint arXiv:2401.02385](#). [arXiv:2401.02385](#) [cs.CL]
- [234] Qiyuan Zhang, Fuyuan Lyu, Zexu Sun, Lei Wang, Weixu Zhang, Wenyue Hua, Haolun Wu, Zhihan Guo, Yufei Wang, Niklas Muennighoff, Irwin King, Xue Liu, et al. 2025. A Survey on Test-Time Scaling in Large Language Models: What, How, Where, and How Well? [arXiv preprint arXiv:2503.24235](#). [arXiv:2503.24235](#) [cs.CL]
- [235] Qingyang Zhang, Haitao Wu, Changqing Zhang, Peilin Zhao, and Yatao Bian. 2025. Right Question is Already Half the Answer: Fully Unsupervised LLM Reasoning Incentivization. [arXiv:2504.05812](#) [cs.CL]
- [236] Shengjia Zhang, Junjie Wu, Jiawei Chen, Changwang Zhang, Zhe Li, Xingyu Lou, Wangchunshu Zhou, Sheng Zhou, Can Wang, and Jun Wang. 2025. OThink-R1: Intrinsic Fast/Slow Thinking Mode Switching for Over-Reasoning Mitigation. [arXiv:2506.02397](#) [cs.CL]
- [237] Dayuan Zhao, Shengcao Cao, Yu-Xiong Wang, and Liang-Yan Gui. 2026. Think in Latent, Explain in Language: Self-Explainable Latent Reasoning. [arXiv:2608.13570](#) [cs.CL]
- [238] Xutong Zhao, Tengyu Xu, Xuwei Wang, Zhengxing Chen, Di Jin, Liang Tan, Yen-Ting, Zishun Yu, Zhuokai Zhao, Yun He, Sinong Wang, Han Fang, Sarath Chandar, and Chen Zhu. 2025. Boosting LLM Reasoning via Spontaneous Self-Correction. [arXiv:2506.06923](#) [cs.CL]
- [239] Xuyang Zhao, Liting Zhang, Zichen Xu, Yong Chen, Wenjia Zeng, Shiwan Zhao, and Qicheng Li. 2026. Latent Thought Credit: Multi-Answer Credit Assignment for Latent Reasoning. [arXiv:2608.01593](#) [cs.CL]
- [240] Zhenyu Zhao, Sander Land, Daniel M. Bikel, and Waseem Alshikh. 2026. Shorthand for Thought: Compressing LLM Reasoning via Entropy-Guided Supertokens. [arXiv:2604.26355](#) [cs.CL]
- [241] Keli Zheng and Zerong Xie. 2025. Synergy: End-to-end Concept Model. [arXiv:2507.12769](#) [cs.CL]
- [242] Zhi Zheng, Yu Gu, Wei Liu, Yee Whye Teh, and Wee Sun Lee. 2025. Soft-GRPO: Surpassing Discrete-Token LLM Reinforcement Learning via Gumbel-Reparameterized Soft-Thinking Policy Optimization. [arXiv:2511.06411](#) [cs.CL]
- [243] Yuyan Zhou, Jiarui Yu, Hande Dong, Zhezheng Hao, Hong Wang, Jianqing Zhang, and Qiang Lin. 2026. LEPO: Latent Reasoning Policy Optimization for Large Language Models. [arXiv:2604.17892](#) [cs.CL]
- [244] 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, et al. 2025. A Survey on Latent Reasoning. [arXiv preprint arXiv:2507.06203](#). [arXiv:2507.06203](#) [cs.CL]
- [245] Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, et al. 2025. Scaling Latent Reasoning via Looped Language Models. [arXiv preprint arXiv:2510.25741](#). [arXiv:2510.25741](#) [cs.CL]