State-Space Search and Transformer-Based AI Hybridizations

Douglas H. Fisher DOUGLAS.H.FISHER@VANDERBILT.EDU
Kyle A. Moore KYLE.A.MOORE@VANDERBILT.EDU
College of Connected Computing, Vanderbilt University, Nashville, TN 37215 USA

Jesse T. Roberts JTROBERTS@TNTECH.EDU
Department of Computer Science, Tennessee Tech University, Cookeville, TN 38505 USA

Abstract

The paper surveys hybrid deliberative (≈symbolic) and reflexive (≈generative) AI systems, focusing on the combination of state-space search, a workhorse of deliberative AI systems, and transformer-based LLMs, an important class of generative AI. We also speculate within the dual-process framing, notably on Type 2, deliberative computation emerging from Type 1, reflexive computation; and introducing PLuMCoT (Primed Lookahead using Multiple Chains of Thought).

1. Introduction: deliberative and generative AI

We are broadly interested in relationships and hybridizations of generative AI and deliberative AI, as are others (e.g., Lenat & Marcus, 2023; Yao, et al., 2023; Kambhampati, et al., 2024).

Deliberative AI corresponds, on the whole, to symbolic AI in the classic tradition, though we prefer the former term, with its emphasis on 'process' rather than 'representation', and because methods that are clearly deliberative are incompletely 'symbolic' (e.g., learning regression trees from data), at least as we understand 'symbolic' as dealing with non-continuous variables, though this is a topic that is undoubtedly ripe for discourse, even now.

A workhorse of deliberative AI is state-space search, traced out through recursive application of *generate-and-test* steps. The generate step can be weak, little informed by constraining or preferential knowledge, necessarily followed by evaluation to decide on what should be pruned, what should be additionally explored, and what is satisfactory relative to a goal or utilitarian ideal. Alternatively, the generate step can be very strong, incorporating knowledge obtained through direct programming by human and/or AI developers, and/or through machine learning (e.g., Langley, 1985; Kambhampati, Stechly, & Valmeekam 2025). Evaluation is still desirable, even with a relatively strong generator, both to guide continued learning of the generator from experience, and also to judge nuance that has not yet been learned, or cannot be learned, because of representational bias in the learning mechanisms.

Deliberative AI has analogs in human intelligence, to what Kahneman (2011) called System 2 thought, largely slow, conscious thought. Turing (1948) and Valiant (2013) informally reference like human processes of *intellectual search* and *mental search*, respectively. Since Kahneman's elaboration of System 2 (and System 1) thinking, there has been some nod in behavioral sciences towards "Type 2" (and "Type 1"), as the latter terms are less suggestive that the thinking is

© 2026 Cognitive Systems Foundation. All rights reserved.

implemented on different substrates (Evans & Stanovich, 2013). We adopt the 'Type' terms. It is Type 2 thought – typically conscious, deliberative, and slow – by human developers, which has traditionally provided the broad strategy designs of deliberative ('Type 2') AI systems.[1]

In contrast, we take generative AI as defined *only by the generate step*, with no, or a trivial test step (Shin, Fisher, & Anderson, 2026). This definition is also consistent with treatment by Kambhampati, Stechly, & Valmeekam (2025). Generative AIs are typically implemented as sophisticated artificial neural network (ANN) architectures, but by our definition they are not necessarily so restricted. Nonetheless, we limit discussion of generative AI to transformer-based models. Generative AI, so limited, does not overtly search, at least not post learning when weights are frozen. Generation is of a probability distribution, P(Y|X), where Y is a possible output "token," and X is a so-called context. Yao, et al. (2023) refer to the generate-only step as strict 'left-to-right' reasoning, by the transformer of an LLM. What 'test' step a generative AI has in this setting is selection of a token from among the tokens covered in the outcome space.[2]

To a pure transformer, a do-until loop, or similar construct, is added to enable a sequence of tokens to be generated, so as to construct a narrative response (**Algorithm 1**). We call this simple algorithmic augmentation to a generative AI, an *iterated generative AI*. If one thinks we are being too pedantic in such distinctions, it is because we hope to generalize and expand simple mechanisms, such as a do-until, into more substantive hybridizations and unifications of deliberative search and transformer-based generation.

Generative AI, by this definition, corresponds roughly to Type 1 – fast, automatic, non-conscious – thought in humans. An iterated generative AI's ability to generate complex narratives stems from machine learning over massive data, which pushes constraining and preferential knowledge from what was an evaluation step during learning, into the generator, a process that is elaborated by Kambhampati, Stechly, & Valmeekam (2025).

```
context ← prompt + background
token ← 'enter'
DO   context ← context +  token
       token ← generate_token (context)
UNTIL token == 'terminate'
```

Algorithm 1: Iterated generative AI.

We recognize that others may mean something different by the term 'generative AI.' Generative systems, for example, are sometimes considered those that learn the joint distribution over input and output, P(X,Y), while discriminative systems are those that learn conditional probabilities, P(Y|X). Adherents of this view reconcile transformer-based LLMs, which explicitly

[1] When we say 'Type 2 AI,' which we occasionally do, we do not mean to imply the 'conscious' characterization, which we reserve for the human side of the analogy.

[2] We have said that (pure) generative AI might have a 'trivial' test step, which we will take as the selection of a token from a transformer implementation's outcome space, based only on each token's generated numeric score (e.g., "conditional probability"). Undoubtedly, 'trivial,' as a characterization of the test step, will change with future work, but undoubtedly it will continue to relate to the depth to which generated output is analyzed.

learn a discriminative conditional probability, to the generative label by showing that repeated, auto-regressive evaluation of the model permits the estimation of the joint probability over all potential completions of the provided input. In this view, generative systems model the transition probability of the state. Generation of an output sequence is then understood as a probability informed walk through the sequence state space.

Another definition that seems adopted by a larger public is that generative AI generates (or produces) content in some domain space, such as fictional or non-fictional narrative, visual art, music, and other creative artifacts. This is indeed a characteristic that might distinguish generative systems, from say classification systems that assign/predict a class for an input, but we should recognize that 'generative' as production of creative content also applies to AI systems long studied in the area of *computational creativity* (e.g., Colton, Lopez de Mantaras, & Stock, 2009; Maher, 2012), which typically have substantial deliberative, symbolic AI components that evaluate their own intermediate and 'final' outputs, and revise those 'outputs' as the system deems appropriate. Generative systems by this popularized definition would include descendants of Meehan's (1977) TALE-SPIN for narrative generation, including interactive narrative (Mateas & Stern, 2003; Reidl & Bulitko, 2013), and Colton et al.'s (2015) The Painting Fool for visual art. But this definition is not one that we use in this paper and we caution it may be misunderstood, or otherwise problematic, in selected other settings for the reasons just mentioned.

In sum, when we speak of *pure generative AI*, we particularly mean (structurally complicated) feed-forward ANN implementations of **generate_token (context)** in Algorithm 1 above, though the complexity of 'token' will undoubtedly vary, even now, and in the future.

2. Hybrid Systems and Architectures

There have been several templates used previously for combining deliberative AIs and ANNs, into hybrid, dual-type AIs. We start, in this section, with the categorization scheme of Anderson & Fisher (2025) to describe selected prior work, followed in Section 3, with a focus on hybrids that stem from overtly combining generative AI mechanisms and state-space search. Each category of hybridization, in one way or another, plays to the respective strengths of the deliberative (symbolic) and generative (neural) AI paradigms.

2.1 Reflex to Reasoning

Mathematics is an area ripe for deliberative/generative hybrid AI systems, because the domain involves what Pólya (1954) characterized as "finished mathematics," largely deductive, and "mathematics in the making," largely inductive/plausible reasoning. The reader will probably have their favorite studies that show that LLMs used for mathematical reasoning (e.g., Mirzadeh, et al., 2024), and other forms of deductive problem solving and planning (e.g., Kambhampati, et al., 2024), are flawed, but LLMs, within a problem-solving context, can usefully serve as generators of ideas to advance an unfolding solution. Google's AlphaProof (Hubert, et al., 2025) is an example of a hybrid AI that uses an LLM component to suggest plausible next steps in a developing proof, and deliberative AI to deductively confirm (or not) the hypotheses.

A template – generative to intelligently guess, deliberative to confirm or otherwise utilize – is intuitive, and increasingly used; Kambhampati's YOCHAN Lab (2026) has done much for the scholarly advance of the proposer-verifier archetype (e.g., Kambhampati, Stechly, & Valmeekam, 2025). LLM-A* (Meng, et al., 2024) uses the template for route planning. In an extensive survey of LLM use in transportation planning, Jin and Ma (2026) enlist the archetype:

> In safety-critical applications, LLMs are best deployed as components within verified architectures rather than autonomous decision-makers. Under the LLM-modulo framework, models act as proposal generators, while external verifiers such as simulators or symbolic controllers validate safety and feasibility. (Jin & Ma, 2026)

Jin and Ma's stance suggests a larger ethical injunction for design and education — we should resist relying exclusively on tuning generative AI to avoid behaviors that we forbid, with no ifs, ands, or buts. Rather, post-generation evaluation and filtering, using logic, which should be taught to humans, is likely the better remedy (Anderson & Fisher, 2025).

Research into *group relative policy optimization* (GRPO) also nicely fits the proposer-verifier framework, in this case for ANN training (Wolfe, 2025), but for outputs that are more than simply single tokens. For every prompt in a training set of size G (e.g., G = 32), a model stochastically generates G outputs. In the "test" phase, a compiler, math verifier, or programmatic checker, in a process referred to as *reinforcement learning with verifiable rewards* (RLVR), tests each output, and assigns a score based on selected criteria. A policy update in GRPO is then made by averaging the test results of the group – outputs that score above the group average have their token pathways reinforced; outputs that fall below the average are suppressed.

Other work adds observational patterns, as knowledge, to concept nets for deliberative commonsense reasoning in systems such as COMET (Bosselut, et al., 2019), CYC (Lenat & Marcus, 2023), and Auto-KG (Ananya, et al., 2024). This too is an area that is growing quickly, with nuanced studies emerging. Zhao, et al., (2025), for example, find an *introspection paradox*:

> while simple introspection tends to benefit weaker models, it often degrades the performance of stronger ones, particularly on simpler tasks. Our deep analysis indicates that this paradox arises from a complex interplay among model capability, task difficulty and the quality of generated knowledge. … To better employ introspected knowledge in LLM, this paper proposes a training-free Adaptive Introspection Strategy that operates in two stages using only the model's internal states (Zhao, et al., 2025)

Long before LLMs, the proposer-verifier archetype was built into human scholarship. Generative AI alone fails from time to time by 'hallucinating' references, but humans often get references wrong initially too. Additionally, in scholarly practice, both humans and LLMs are known to confound ideas and arguments, and to misattribute ideas to a wrong source, and/or to give an incomplete set of sources. Importantly, humans follow generative steps by confirming, correcting, and adding references – they reflect, including through peer review, and revise, as needed. Hybrid, dual-type AIs can go through similar processes (e.g., Kambhampati, Stechly, &

Valmeekam, 2025). In generation-augmented retrieval (GAR), as opposed to the more commonly used RAG, a generated string, such as a bibliographic reference, is followed by retrieval(s) to either confirm the reference, or reject and revise, and to look into additionally retrieved references (Anderson & Fisher, 2025). Long (2023) recognizes the same kinds of limitations of generative AI, which humans mitigate through deliberation, in his own motivation for hybrid AIs:

> To ensure correctness, a good practice for a human solver is to carry out verification procedures at *every step* of the problem-solving process, thereby ensuring the credibility of the final solution. In comparison, auto-regressive language models do not explicitly perform logical correctness checks as it generates a new token based on the previous tokens. This limits the model's capacity to rectify its own mistakes. A minor error could be amplified as the model generates more tokens, thereby leading to rapid solution quality deterioration and making it difficult to recover from mistakes. (Long, 2023)

Indeed, there is much reason to believe that hybrid AIs of the future will be super scholars, biographers of ideas for scholarly communities that perform better than the best human scholars (Shin, Fisher, & Anderson, 2026) – the real problem will be to constrain and organize the super scholar's expression to avoid overwhelming the human reader. The very topic of this paper, fast developing AI hybrids, but with old intellectual roots, is ideal for thinking about an AI super scholar, its knowledge representations, and its strategies for composition and communication.

2.2 Reasoning and Communication

Another template for AI hybrids is using deliberative AI to explore, through search, various strategic options, as in the case of Meta's Cicero system (Meta Fundamental AI Research Diplomacy Team, et al., 2022) for playing to the social game of Diplomacy, and then using generative AI to communicate with other players on the strategic choices, and to negotiate agreements, again with deliberation on the backend of these negotiations.

2.3 Setting the Stage

The 'Reflex to Reasoning' systems illustrate how generative AI can give outputs that are then subject to various forms of deliberative reasoning – the former sets the stage for the latter, so to speak. The directionality can be switched too, with retrieval-augmented generation (RAG), being one example (Lewis, et al., 2021).[3] A different form of deliberation-to-prepare-for-generation is neuro-symbolic research that uses (evolutionary) search for ANN network configurations (Stanley, et al., 2019; Yao & Liu, 1997), the 'best' of which are then used for learning.

[3] We consider the *retrieval* step in RAG as deliberative, even if there might be debate on whether it qualifies as *AI* per se. Our reasoning is that retrieval employs sophisticated information structures, selects sources based on logically-expressible criteria and scoring functions, and is referenced in introductory AI (e.g., Poole & Mackworth, 2023), and so represents a form of AI, even if a simple one (Anderson & Fisher, 2025).

3 Generic Search and LLM (transformer) generation

Having surveyed some generative-deliberative hybridization strategies, we now expand on hybrid forms that specifically combine state-space search and LLM generation (or decoding). LLM generation can be viewed as an instantiation of generic search, and LLM generation can serve to implement components of generic search (e.g., Long, 2023; Yao, et al., 2023; Besta, et al., 2024).

Algorithm 2 defines a generic search procedure. It is not defined as goal search only, but as a broader utility search, and one that does not stop at the first satisfactory state found. As noted earlier, search is defined by multiple generate-and-test paired steps, highlighted by bold text for determining **children** and **scored_states**.[4] The **PriorityQueue**, **solutions**, holds complete solutions, organized by a quality score. The **PriorityQueue**, **fringe**, contains search states awaiting expansion through **generate_successors** (aka 'children').

The function, **generic_search**, generalizes many standard search algorithms, both uninformed and informed (e.g., heuristic), including an *anytime search*, with the addition of the **solutions** priority queue that can be interrogated any time (with a small algorithmic tweak). We assume that the **PriorityEnqueue** operation is implemented to appropriately manage multiple paths to same states, as well as perform other niceties. Further, we assume the definition of a state includes a field in which **generate_successors** places important path information.

```
FUNCTION generic_search(start):
   PriorityQueue solutions ← {}
   PriorityQueue fringe ← {[score(start), start]}
   WHILE NOT stop? AND fringe IS NOT empty DO
      [score, current] ← PriorityDequeue (fringe)
      IF terminal_state(current) THEN
         PriorityEnqueue([score, current], solutions)
         stop?()   // query on whether to stop searching
      ELSE  // generate and test
         children ← generate_successors(current)
         scored_states ← {[score(c), c] | c in children}
         FOR each [s, c] in scored_states DO
            PriorityEnqueue([s, c], fringe)
   RETURN solutions
```

Algorithm 2: Generic search.

Search, with generate-and-test, is the workhorse of deliberation in "classic" AI. Shortly after the generative AI boom that came with the release of LLMs, work on embedding generative AI within state-space search emerged from multiple projects (e.g., Long, 2023; Yao, et al., 2023). It is worth adding too, in part because it signals what's to come, that in the collaboration between a human user and an AI platform with a dominant generative component, it is the human that is

[4] The proposer-verifier archetype of Section 2 is one instantiation of generate-and-test in state-space search.

doing much of the deliberation, undoubtedly through an informal and non-systematic search procedure, say with contexts (including prompts) as the state descriptions, perhaps across all previous conversations between the human and the AI. While the human user's memory is far from a strict priority queue, it is that memory that directs much of the search, though this deliberation is presumably being displaced rapidly by AI platforms with increasingly powerful deliberative capabilities built into the AI side of human and AI collaborations (e.g., see Kambhampati, Stechly, & Valmeekam 2025).

3.1 LLM Generation as Greedy Search

Now consider **llm_gen**, a near-minimal transformer-based LLM shown in **Algorithm 3**, which further details the iterated generative AI of Algorithm 1. In Algorithm 3, **llm.forward** is the feedforward inference procedure of the ANN-based LLM. This function takes a text sequence as **context** and returns an activation level for each token in the model's vocabulary (**logits**). These **logits**, after optional filtering, are converted to a probability distribution using a **softmax** function. Softmax optionally takes a temperature term (**temperature**) that modifies the uniformity of the probability distribution; the distribution approaches uniform as temperature approaches infinity; in contrast, a temperature of zero places all probability mass in the most probable token. *Filtering*, which removes all activation from a subset of tokens to force their generation probability to zero, can be defined arbitrarily, but most commonly uses *top-p* and/or *top-k* filtering. Top-p, which is typically performed first due to its internal softmax call, removes the maximum number of tokens such that the total probability mass does not fall below p, therefore yielding the smallest credible set of tokens with combined mass of at least p. Top-k is more straightforwardly defined, removing all but the k most probable tokens from consideration.

```
FUNCTION llm_gen (llm, context, max_length, k=|V|, p=1.0, temperature=1.0):
  response ← []
  resp_length ← 0
  WHILE NOT stop? do
    logits ← llm.forward(context)
    filtered_logits ← filter(logits, k, p)
    filtered_probs ← softmax(filtered_logits, temperature)
    token ← select(filtered_probs)
    response ← response + token // '+' is append
    context ← context + token
    response_length ← response_length + 1
    IF token == terminate OR response_length >= max_length
    THEN stop? ← True
  RETURN response
```

Algorithm 3: Computing **logits** through **token** expand **generate_token** in Algorithm 1.

After filter and softmax transformation, the next generated **token** is **select**ed from the resulting probability distribution, typically using either greedy (i.e. deterministic, *probability maximizing*) selection or *weighted random sampling* (i.e. stochastic, *probability matching*). The process repeats autoregressively, with the newly generated token appended to the **context**, until the termination criterion is achieved. The termination criterion is, most typically, that a special **terminate** token is selected or a maximum **response_length** is reached.

LLM generation can be viewed as a special case of generic search, in which the state is the current context (input context and partial response) and the generate_successors function is the forward-filter-sample sequence, allowing token filtering and softmax temperature to balance output stability and creativity. The selection process implicitly forces the fringe to be size one throughout, effectively rendering this process as a *hill climb*. Each next step of the hill climb is decided by a score (i.e., probability) weighted sample over the filtered token set.

We can generalize llm_gen with a larger, explicit fringe, and to enable multi-output response with a 'solutions' priority queue. In so doing, this section's observation that LLM generation instantiates generic_search (as a hill climb with states represented as strings) will morph into a larger possibility — that a generalization of LLM generation can unify more fully with generic_search. This unification, assumes something else, which is that just as generic_search can be implemented with states of different formats, llm_gen can be so generalized as well. Importantly, theory of formal languages and computation (Hopcroft & Ullman, 1969) suggests that with strings as states, and therefore formal languages as state spaces, this eventuality is certainly possible. Thus, in what follows, we continue to talk of LLM generation of strings (rather than states per se), but from a broader class of languages than natural languages and programming languages, with no loss of theoretical generality.

Yao, et al. (2023), Long (2023), and Xie, et al. (2023) recognized that LLM generation traced out one path of what could be a larger search. Long (2023) implemented a backtracking, 'depth-first' search; Yao, et al., (2023) explored breadth-first and depth-first searches; and Xie, et al. (2023) implemented beam search. These works addressed step size, so that rather than having an LLM necessarily generate a 'complete' narrative derivation, smaller segments can be stochastically generated, then evaluated, and enqueued, as appropriate to the search strategy. These segments have been called 'thoughts' and 'ideas' elsewhere, but we seek to mitigate anthropomorphism; there might also be relationships with historic understanding of 'chunks' in cognitive science that are worth exploring. Besta, et al. (2024) generalizes further, implementing search of directed acyclic graphs of LLM-generated segments. These general strategies constitute 'reasoning,' in which the search of possible outcomes is known as "test-time inference" or "test-time compute" in large *reasoning* models (LRMs), in contrast to assuming that all knowledge has been pushed back into the generator (Kambhampati, Stechly, & Valmeekam 2025).[5]

Other important generalizations of this framework move from search-as-reasoning, in the language of LRMs and cognitive architectures (e.g., Langley, Laird, & Rogers, 2009), into

[5] It would be interesting to explore the relationship of 'test-time inference' in LRMs with the historic concept of 'lazy learners' in the machine learning literature (e.g., some case-based and instance-based learning approaches). Looking at these literatures together suggests a *continuum of laziness* that may prove a useful means of characterizing systems.

related, but distinguishable cognitive capabilities of 'planning' and 'acting' (Hao, et al., 2023; Zhou, et al., 2024). In this context, Hao, et al. (2023) uses LLMs for world modeling as well.

Finally, we briefly recognize research that was prescient in important respects. In 'Connectionist Problem Solving,' Hampson (1990, 1994) sought to bridge ANNs and deliberative reasoning, arguing that strictly-neural architectures supported stepwise problem solving. Hampson focused on abbreviated search with one-step lookahead, consistent with his interests in animal behavior, but his conceptualization of building action sequences to achieve goals maps onto today's search-enabled ANN-based reasoning models, a topic for another paper.

3.2 LLM Generation for Implementing Components of Search

In contrast to instantiation/unification above, the perspective here is that LLM generation can implement various choice points of generic search; this provides scaffolding for further specifics.

3.2.1 Generating Successors

An obvious point where generative AI has been applied, in effect, in systems already surveyed, is generate_successors of Algorithm 2. LLM generation (i.e., Algorithm 3) generates one successor, and multiple calls would generate multiple successors, where each call to llm_gen could select the next token by a stochastic *probability matching* (aka random weighted sampled) strategy (among the top k), for example, rather than a *probability maximizing* strategy.

In state-space search over discrete, finitely-branching domains, generate_successors often enumerates all valid immediate successors as determined by the internal (if implicit) structure of the task itself. The llm.forward function, as the analog to generate_successors, falls under this umbrella in that the successors of a state, s, are states, s', such that $s' \in \{s + v \mid v \in Vocab\}$, where '+' is the append operation. Tasks that don't satisfy the finite constraint on successors will require generation of a promising subset of the total successor sub-space. Generative AI, particularly when invoked repeatedly as a distinct sub-process, may be a promising successor generator for complex tasks of this form, though there are many variations that do not easily fit generate_successors as just described. Here are three noteworthy examples.

Mixture-of-expert (MoE) models include multiple LLM variations, acquiring a separate gating mechanism to determine the most appropriate variation to employ in generate_successors; this conditional selection can be placed into generate_successors itself.

Obscured reasoning, with some model responses hidden through tags, is an alternate generate_successors, but unlike MoE, this alternate typically executes only at the beginning of the response, with resulting reasoning traces often discarded between user queries.

Tool calls invoke external, typically non-LLM, mechanisms to supplement reasoning (e.g., a mathematics or logic solver, with the returned answer being inserted into the response).

A model will choose the process to generate successors at each inference step, though the complexity of possible configurations begs more decomposition, which we do not address here.

3.2.2 The Score Choice Point

Most canonical search algorithms are defined, in part, by their scoring function form (e.g., Poole & Mackworth, 2023). Examples are *Dijkstra's search*, *greedy best-first search*, and *A* search*, distinguished by whether 'score' is defined respectively, by path-thus-far cost, expected (heuristic) distance from the goal, or both. Importantly utility-maximizing searches can use measures besides 'distance'. Depth-first (DFS) and Breadth-first (BFS) searches, use stacks and FIFO lists, respectively, but equivalently, they can use appropriately configured priority queues. In llm_gen, 'score' is best understood as the probability assigned to each allowed next token.

The possible applications of generative AI in scoring are many, including mechanisms where AI is prompted to directly assess the quality of a state (e.g. "On a scale of 1-100, how would you judge state < s > for task < t > with goal <g>?"). There is also growing literature on *self-assessment uncertainty quantification* as a basis of scoring (Kadayath, et al., 2022; Lin & Evans, 2022; Xiong, et al., 2024). Contextually-informed scoring will also be important (e.g., execution of a unit test suite for a code-generation task, logic and mathematics solvers). Lookahead scoring, described more in the following section, could generate Monte Carlo tree search-like rollouts that take likely future generation steps into account. This possibility echoes prior work by Hao, et al. (2023), in which LLMs are inserted into a MCTS as *world models* that facilitate versatile and multi-task simulation during the rollout phase of the overall MCTS-based planning algorithm. Zhou, et al. (2023) further iterate on this by integrating LLMs into node expansion and initial node scoring. Gurnee, et al. (2026) suggests that models have, in a sense, begun naturally developing such a lookahead scoring ability, as described shortly. These variations are not necessarily disjoint, as illustrated by the *APriCoT* system (Moore, et al., 2025), explained shortly, which utilizes self-assessment direct scoring after a limited lookahead.

3.2.3 Other Choice Points

Less common choice points vary across llm_gen and generic_search implementations.[6] **Queue Management**, is one example, with successors typically enqueued and dequeued based exclusively on the scores assigned to states. The biggest departure in enqueue are beam searches, which potentially truncate the fringe after enqueues. Variation in dequeue is largely restricted to the choice between greedy (probability maximizing) and random weighted sampled (probability

[6] Another intriguing possibility is using generative AI for implementations at meta-choice points, such as 'configuration selection.' If the generative AI is 'unfrozen,' subject to continued training, then this fits into the larger area of meta-learning, an old subfield of machine learning (e.g., Brodley, 1994; Chan, 1996; Gordon, 1990; Vilalta & Drissi, 2002). In "Setting the Stage" above, we noted that deliberation in the form of evolutionary search could be used for selection of a best ANN design, but using Type 1 to inform Type 2 is also possible. For example, tasks that can be decomposed into structurally distinct subtasks (e.g., a code-generation phase of an otherwise conversational LLM response) may benefit from per-iteration variation on the implementation selected for the other choice points. Most obviously, this might involve variation on how the current set of successors (both at token level and especially at sequence level) are scored. This is an essential aspect of PLuMCoT, described shortly, which recognizes that the score that a lookahead should be judged on may be very context dependent, such as to what the current goal is, with there being near-continuously varying trade-offs between logical coherence, syntactic adherence, creativity, alignment compliance, etc. In principle, this idea of meta-choice can be extended to all choice points as well.

matching) selection. The remainder of meaningful variation is in tie-breaking between states, which is common in CSPs (e.g., Minimum Remaining Values, Least Constrained Value, and Degree heuristics), and in other searches to break ties using the heuristic component of scores or some aspect of state novelty, for example. Generative AI systems might serve in tie-breaking mechanisms, particularly in the case of complex tasks with vaguely or informally-defined goals.

Successor Filtering is a choice point tasked with reducing the number of successors identified by generate_successors to a more manageable target set such that the successors that survive filtering are those that are considered most promising for future exploration. State-space search rarely involves an explicit filtering step, with the closest common analog being the selection of a finite successor set for state spaces with infinite branching factors. There are exceptions, however, such as filtered beam search (Ow & Morton, 1988; Mejía & Niño, 2017), which applies a cheap filtering step to mitigate expensive heuristic evaluation, and to enforce search tree diversity. Conversely, this stage is ubiquitous in LLMs, usually in the form of top-k and top-p (or nucleus) filtering, described earlier. In the LLM case, filtering of this sort is especially important when the Queue Management choice point is based on random weighted sampling (probability matching) to avoid selection of marginally most probable successors. Since filtering in this case is based on the token probabilities, filtering and scoring may be interleaved such that scoring occurs both before and after filtering and both may, conceivably, be successively applied to iteratively narrow down candidate successors.

The **Termination Criteria** choice point is the last one that sees significant variation in search. Most commonly, this is a goal checking function that determines whether the current state possesses all desired attributes. Generative AI systems may be particularly useful for nebulously defined goals, such as in narrative generation, where LLM judgements may be imperfect, but nonetheless quite effective. In anytime searches, said goal-checking only determines when solutions are added to the solution set (i.e., 'solutions' in generic_search), but search does not terminate until a higher-order termination criterion is met (e.g., user interruption, time-limits, sufficient utility, maximum solution limit).

In sum, each of the choice points in generic_search can utilize generative AI in their implementation. Further, as llm_gen is a hill-climbing instantiation of generic search, so the same applies there — application of llm_gen to choice points in llm_gen, which may be recursively applied, thus rendering the possible hybrid search implementations seemingly unbounded.

4 PLuMCoT as Hybrid AI

PLuMCoT (Primed Lookahead using Multiple Chains of Thought) is a **proposed** hybrid AI framework that is intended to improve reasoning, flexibility, and customizability. It is a generalization of the **existing** APriCoT (Agnostically Primed Chain-of-Thought) system (Moore, et al., 2025). APriCoT is a variation on chain-of-thought for answering multiple choice questions. In particular, APriCoT generates a separate reasoning trace for each of the candidate answers. These reasoning traces are generated using the standard llm_gen function by adding instructions to justify an associated candidate answer. After reasoning about all answers, the model self-assesses its confidence in each. The chosen answer is the one that yields the strongest positive

response after justification. APriCoT improves answer accuracy and reduces heuristic bias compared to standard chain-of-thought.

```
FUNCTION plumcot(llm, context):
    response ← []
    WHILE True DO
        IF stop_criteria(context, response)
           RETURN response
        ELSE
            IF trigger? (context) THEN
                logits ← llm.forward(context)
                filtered ← filter(logits)
                scores ← {}
                FOR each candidate in filtered DO  // in parallel
                    L ← llm_gen(context + candidate)
                    s ← score(context + candidate + L)
                    scores.add((s, candidate))
                selected ← select(scores)
            ELSE selected ← llm_gen(context)
        response ← response + selected
        context ← context + selected
```

Algorithm 4: Pseudocode for PLuMCoT (Primed Lookahead using Multiple Chains of Thought) Adapted and simplified from Moore (2026).

Unlike APricoT, PLuMCoT (**Algorithm 4**), is intended to answer queries through free generation of textual responses. PLuMCoT uses lookahead: after filtering to a select subset of candidate completion tokens, the standard llm_gen is used to generate a further sequence of tokens,[7] continuing after each candidate token; these continuations are to be effectively done in parallel. The continuations approximate where each generation is likely to continue if the associated token is chosen at the current iteration. Each lookahead is subjected to a scoring function to determine which candidate token leads to the most preferred expected continuation. From this point forward, search continues as in the standard llm_gen function, selecting (either greedily or via sampling) from the candidate tokens based on these lookahead scores, appending the selected token to the context, and repeating the process at the subsequent iteration.

Base PLuMCoT is expected to provide generation quality benefits because the testing phase of generate-and-test can incorporate expected future information. By expanding score functions, to include checking output for logical coherence, factual correctness, alignment compliance, etc.,

[7] Desirable lookahead lengths will vary, and lookaheads are not necessarily intermediate reasoning traces. In light of the J-lens work referenced shortly, we intuit that fewer tokens, on the order of 10-20, are probably sufficient for most tasks, with code generation likely being a common exception to our intuition.

PLuMCoT can prune choices that likely lead to suboptimal paths early, and discourage "locking-in" to generation paths from which the model might "struggle" to recover.

Benefits will likely accrue when lookaheads are scored by smaller, more efficient auxiliary discriminative (or regression) models along various evaluation dimensions. For an auxiliary alignment verification model (such as one that recognizes the presence of violent, offensive, or otherwise undesirable content), this can give the system a sort of "alignment reasoning" that does not strongly rely on the effective alignment training of the underlying LLM. This use case is supported by existing work in the space of decoding-time alignment, in which LLM alignment protocols are enforced in frozen base models without learned weight updates. Noteworthy examples include Li, et al. (2024), who implement a rewind mechanism rather than a lookahead, and Mudgal, et al. (2024), who use inference-time scoring of the context and candidate tokens to control generation. This means that newly released, unaligned models can be inserted into the PLuMCoT algorithm and reasonably expect similar alignment compliance without expensive, possibly redundant (across other model releases) fine-tuning. This further allows for user choice and customizability, as differing alignment protocols for differing user bases and task sets may be achievable by simply swapping out the auxiliary scoring model.

PLuMCoT relates to the various approaches surveyed in Besta, et al. (2024), including Trees of Thought, introduced earlier. PLuMCoT can be viewed also as a form of CoT-SC as Besta describes it, but CoT-SC, as described in its original paper (Wang et al 2023), explicitly refers to "best" candidate generation as based on consistency across all the chains of thought, whereas PLuMCoT is planned as unconstrained, using an arbitrary choice of relative scoring function.

CoT-SC, alongside other methods surveyed in Besta, et al (2024), is primarily concerned with the choice of final answer, that is the summarized answer provided after chain of thought (e.g. "… Therefore the correct answer is X"). PLuMCoT differs here by using a comparative approach for all token generation steps, rather than restricted to a privileged "thinking" phase.[8]

Self-Evaluation Guided Beam Search for Reasoning (Xie, et al., 2023) appears similar to PLuMCoT. The biggest difference is that the former commits the entire lookahead after scoring rather than only the initial token. This is almost certainly more efficient, but it is less flexible, as the model may be locked in to suboptimal reasoning paths by a relatively arbitrary choice of which iteration should be the anchor for the lookahead. That is, after committing token t_k, it may turn out that token t_{k+2} is also an important branching point for guiding the reasoning, but this is missed because tokens t_k through t_{k+n} (where $n > 2$) are all determined solely from t_k.

PLuMCoT also bears some similarity to existing methods in the controlled decoding literature. Wang and Zhou (2024), for example, explores CoT decoding at numerous generation steps using a process that can be reasonably cast as an instantiation of PLuMCoT that uses top-k filtering, deterministic lookahead, and self-reported confidence-based scoring. Similarly, Neurologic A*esque (Lu, et. al., 2022) uses lookaheads scored by their overall likelihood of satisfying explicit constraints to guide generation. Both of these methods show improved reasoning abilities by identifying instances in which lower probability tokens lead to more preferred future

[8] It's worth noting that PLuMCoT might be adapted to enrich the search paths in CoT-SC with additional pseudo-heuristic information from the lookahead scoring.

responses. PLuMCoT serves largely as a generalization of these, as well as the aforementioned APriCoT, covering all variations on a shared generate-lookahead-test format.

PLuMCoT also has similarities to techniques employed in model predictive control (MPC), Monte Carlo tree search (MCTS), and reinforcement learning with verifiable rewards (RLVF). In MPC terms, PLuMCoT is structurally classed as an instance of receding-horizon control, wherein, at each timestep, candidate actions are simulated over a finite horizon after which the most promising action is selected and committed. More specifically, it exists within the truncated rollout scheme of Bertsekas (2022), if the autoregressive decoder is taken as the base policy and the score function as a terminal cost approximator. Scoring the outcomes of multiple lookaheads from a shared starting state is also central to Monte Carlo tree search, where default-policy rollouts estimate board-state values (Kocsis & Szepesvári, 2006; Browne, et al., 2012), and to GRPO (Shao, et al., 2024), where responses sampled from the current policy for the same prompt are scored and each response's advantage is computed relative to the group. GRPO is currently widely used in alignment fine-tuning using RLVR (Lambert, et al., 2024), where each response is rewarded by a scoring mechanism suited to its prompt, echoing the context-dependence stressed in the earlier discussion of PLuMCoT's scoring function.[9]

It is likely that many generation steps will not substantially benefit from PLuMCoT's computationally-expensive lookahead evaluation. As such, PLuMCoT would benefit by gating its application with a trigger function that may most simply be implemented as an uncertainty threshold, wherein the model's uncertainty regarding the next token to be generated determines whether a lookahead evaluation is pursued. This addition links PLuMCoT to dual-process theory. The model's default llm_gen response is, under this view, the automatic Type 1 response that is contextually subject to a negotiation between necessity and available computational resources, overridden at times by a more deliberative Type 2-like process.

5 How Might Type 2 Emerge from Type 1?

We are interested in the general interplay between Type 1 and Type 2 processing in a computational context — in AI; analog considerations in humans and animals are fraught with debate in cognitive and evolutionary psychology, which we do not address here.

A transformer-based LLM is primarily generative and thus Type 1 in nature. However, as we have assumed throughout this paper, there is no architectural or mechanistic limitation that precludes evaluation of generative outputs. Indeed, as already noted, learning a strong generator assumes that Type 2 consistent evaluation behavior enables the backing up, from test phase to generate phase, of preferential and hard constraint knowledge. This knowledge can be quite complex, to include a litany of training targets beyond straightforward next token prediction, so as to also ensure compliance with social norms, to follow instructions, to take verbal reasoning actions to improve answer quality, and to improve contextual relevance (Scott & Roberts, 2026).

[9] There is a potentially interesting line of thought here that PLuMCoT is an attempt at moving GRPO into inference, rather than training, and that it can yield similar, though possibly reduced, benefits as the fine-tuning without the up-front expense. This could be argued to introduce (though not novel in the broader world of RL) a trade-off between offline (GRPO) policy improvement and on-line (PLuMCoT) policy guidance.

Learning a strong generator through reinforcement strategies, such as RLME (Rentschler & Roberts, 2026) and GRPO (Shao, et al., 2024) seeks to maximize the probability of arriving at rewarding conclusions, illustrating Type 1 capabilities improving from Type 2 evaluation, a view common in history of machine learning of search control (e.g., Langley, 1985).

But what of Type 2 emerging from Type 1? For example, what of models that come to self-reflect during generation so as to correct logical failures and ill-fit solutions? How might self-reflection and revision emerge from generation only?

While the search-based strategies that we have surveyed are intended to anticipate different outcomes in a search space of generative outcomes, and choose accordingly, a single iterated-generative process (Algorithm 1) can, in theory, non-systematically explore parts of this search space in one continuous hill-climb. This is a view intuited from similar framing of incremental machine learning that hill-climbs through a space of classifiers (e.g., Fisher & Pazzani, 1991). Our thought experiment of an unencumbered iterated-generative process will make mistakes along the way, which it cannot erase later,[10] but which it can correct later, and add that correction to the growing context, before continuing on. What can be learned from such a derivation when annotated by external feedback of positive/correct and negative/undesirable signals? Among the possibilities, we imagine 'chunks/macros.'[11]

Importantly, the underlying structure on which most LLMs are developed, the decoder-only transformer (Radford, et al., 2019) is Turing Complete (Roberts, 2024). Specifically, the transformer is a form of Wang B machine, which is a Turing machine incapable of deleting tape content, and yet it is still as computationally expressive as the basic TM.

While the class of model to which LLMs belong is computationally universal, any specific LLM instance is unlikely to be, as they are trained for next token prediction rather than general computation. That is, they are trained to predict the probability of the next token, $P(T_{t+1} \mid C_t)$, given the preceding tokens/context $C_t = [T_0, \ldots, T_t]$. Interestingly, when the subsequent token, T_{t+2}, is to be generated the context is $C_{t+1} = [C_t, T_{t+1}]$. While T_{t+1} is doubtlessly important to the generation of T_{t+2}, much of the information which drives its selection is derived from C_t. So, it is reasonable to expect that probable tokens for T_{t+2} are foreshadowed in latent representation in selecting T_{t+1}, which is precisely what is found when latent representations are evaluated through the Jacobian analysis process called J-lens (Gurnee, et al., 2026).

Perhaps next token prediction alone is a sufficient learning target to encourage a model to develop internal representations of likely, slightly more distant future tokens yet to be generated. An important question is whether the awareness of likely future tokens, say T_{t+2}, impact the present generation of T_{t+1}. A model informed by a prediction over future tokens essentially possesses a latent plan, or at least a latent chunk. If an LLM were to build such a latent plan, and use it to inform the selection of next token, then presumably a significant latent test capability has emerged within the model for the purpose. However, it is not currently known whether the more distant future tokens would be causal or correlative with respect to the resulting next token.

[10] We imagine that expanding token vocabulary, to include erasure tokens, and token placement, not simply appending, will enable generative AI designs that can substantively revise contexts.

[11] We also imagine the learning of 'goto,' at least in effect, though we do not elaborate here.

More research is needed to understand how Type 2 AI might emerge from Type 1. Inversely, it is likely that many concepts of AI design and development will continue to draw from Type 2 AI. We imagine a future where deliberative mechanisms within hybrid AIs will be 'compiled' entirely into ANNs (Anderson & Fisher, 2025), and that understanding and use of strictly-connectionist problem solving architectures (e.g., Hampson, 1994) will be important.

6 Concluding Remarks

LLMs, and other generative AI systems, are being pushed in research and engineering thrusts to engage in reasoning-like behavior. Reasoning has traditionally been achieved in AI using search, which LLMs are not generally seen as performing. Nonetheless, properly viewed, standard LLM inference loops (llm_gen) are a highly greedy instantiation of a state-space search. This can be taken as a claim that they are reasoning in a Type 2 sense, but in a very limited fashion.

By recognizing a larger possible unification, we identify other ways that LLM reasoning can be modified with state-space search and where LLMs can be included as components within a state-space search, allowing for a full realization of Type 2 reasoning with seemingly Type 1 LLM inference as a driving engine. We have surveyed work to date in this area, including our PLuMCoT proposal, and touched on implicated ethical issues, notably safety and scholarship, which deserve much greater attention.

Early results have also given evidence that LLMs, by virtue of training with limited Type-2 reasoning, may have learned to implicitly plan multiple tokens ahead, therefore suggesting they might learn to touch Type 2 reasoning within latent space, an observation that suggests subsequent, ambitious investigation.

Acknowledgements

We thank Professor Helen Shin and the Cognitive Systems reviewers for important and impactful comments on the initial submission. Authors are listed alphabetically, with equal contribution.

References

Ananya, A., Tiwari, S., Mihindukulasooriya, N., Soru, T., Xu, Z., & Moussallem, D. (2024). Towards Harnessing Large Language Models as Autonomous Agents for Semantic Triple Extraction from Unstructured Text. *CEUR Workshop Proceedings, 3747*. Hersonissos, Greece.

Anderson, C. B., & Fisher, D. H. (2025). *Artificial Intelligence for Academic Libraries*, New York, NY: Routledge.

Bertsekas, D. P. (2022). *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*, Belmont, MA: Athena Scientific.

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., Hoefler, T. (2024). Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *AAAI Conference on Artificial Intelligence (AAAI-24)*, 17682-17690. https://ojs.aaai.org/index.php/AAAI/article/view/29720/3123.

Bosselut, A., Rashkin, H., Sap, M., Malaviya, C., Celikyilmaz, A., & Choi, Y. (2019). COMET: Commonsense Transformers for Automatic Knowledge Graph Construction. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1470.

Brodley, C. (1994). Recursive Automatic Bias Selection for Classifier Construction. *Machine Learning 20*, 63-94.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games, 4(1):1–43.

Chan, P. K. (1996). An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning. Doctoral dissertation, Graduate School of Arts and Sciences, Columbia University, New York, NY.

Colton, S., Halskov, J., Ventura, D., Gouldstone, I., Cook, M., & Pérez Ferrer, B. (2015). The Painting Fool Sees! New Projects with the Automated Painter. *Proceedings of the International Conference on Innovative Computing and Cloud Computing*.

Colton, S., Lopez de Mantaras, R., & Stock, O. (2009). Computational Creativity: Coming of Age. *AI Magazine*, *30*, 11–14. https://doi.org/10.1609/aimag.v30i3.2257

Evans, J. S. B., & Stanovich, K. E. (2013). Dual-process theories of higher cognition: Advancing the debate. *Perspectives on psychological science*, *8*, 223–241.

Fisher, D., & Pazzani, M. (1991). "Computational Models of Concept Induction," in Fisher, D., Pazzani, M., & Langley, P. (eds.), *Concept Formation: Experience and Knowledge in Unsupervised Learning*. San Mateo, CA: Morgan Kaufmann, 3–44.

Gordon, D. F. (1990). Active Bias Adjustment for Incremental, Supervised Concept Learning. PhD Thesis, University of Maryland.

Gurnee, W., Sofroniew, N., Pearce, A., Piotrowski, M., Kauvar, I., Chen, R., Soligo, A., Bogdan, P., Ong, E., Wang, R., Thompson, B., Abrahams, D., Kantamneni, K., Ameisen, E., Batson, J., & Lindsey, J. (2026). Verbalizable representations form a global workspace in language models. https://arxiv.org/abs/2607.15495.

Hampson, S. (1990). *Connectionist Problem Solving*. Boston: Birkhauser.

Hampson, S. (1994). Problem Solving in Artificial Neural Networks. *Progress in Neurobiology 42*, no. 2, 229–281. https://doi.org/10.1016/0301-0082(94)90065-5.

Hao, S., Gu, Y., Ma, H., Hong, J., Wang, Z., Wang, D., & Hu, Z. (2023). Reasoning with Language Model is Planning with World Model. *Conference on Empirical Methods in Natural Language Processing*, 8154–8173. https://aclanthology.org/2023.emnlp-main.507/.

Hopcroft, J. E., & Ullman, J. D. (1969). *Formal Languages and Their Relation to Automata*. Addison-Wesley.

Hubert, T., Mehta, R., Sartran, L. et al,. (2025). Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature, 651,* 607–613.

Jin, Y., & Ma, J. (2026). A survey of large language models in transportation planning: modelling, design and decision-making. *Transportmetrica A: Transport Science*, 1–46. https://doi.org/10.1080/23249935.2026.2631152.

Kadavath et al. (36 authors) (2022). "Language Models (Mostly) Know What They Know." https://arxiv.org/abs/2207.05221.

Kahneman, D. (2011). *Thinking, Fast and Slow*. Macmillan. ISBN 978-1-4299-6935-2.

Kambhampati, S., Stechly, K., & Valmeekam, K. (2025). (How) Do reasoning models reason? *Annals of the New York Academy of Sciences*, 1547(1), 33–40. https://doi.org/10.1111/nyas.15339.

Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, P., & Murthy, A. (2024). Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. *41st International Conference on Machine Learning* (pp. 22895-22907). Vienna, Austria: https://proceedings.mlr.press/v235/kambhampati24a.html.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *European Conference on Machine Learning*, 282–293.

Lambert, N., et al. (23 authors) (2024). "Tulu 3: Pushing frontiers in open language model post-training." *https://arxiv.org/abs/2411.15124*.

Langley, P. (1985), Learning to Search: From Weak Methods to Domain-Specific Heuristics. *Cognitive Science, 9*: 217–260. https://doi.org/10.1207/s15516709cog0902_2.

Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2), 141-160. https://doi.org/10.1016/j.cogsys.2006.07.004.

Lenat, D., & Marcus, G. (2023). "Getting from Generative AI to Trustworthy AI: What LLMs might learn from CYC." https://arxiv.org/abs/2308.04445.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., et al. (2021). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." https://doi.org/10.48550/arXiv.2005.11401.

Li, Y., Wei, F., Zhao, J., Zhang, C., & Zhang, H. (2024). Rain: Your language models can align themselves without finetuning. *International Conference on Learning Representations*, 20580–20595.

Lin, S., Hilton, J., & Evans, O. (2022). "Teaching Models to Express Their Uncertainty in Words." https://arxiv.org/abs/2205.14334.

Long, J. (2023). "Large Language Model Guided Tree-of-Thought." https://arxiv.org/abs/2305.08291.

Lu, X., Welleck, S., West, P., Jiang, L., Kasai, J., Khashabi, D., Le Bras, R., Qin, L., Yu, Y., Zellers, R., et al. (2022). Neurologic a* esque decoding: Constrained text generation with lookahead heuristics. *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 780–799.

Maher, M. L. (2012). Computational and Collective Creativity: Who's Being Creative. *Proceedings of the International Conference on Computational Creativity*, 67–71. https://computationalcreativity.net/iccc2012/wp-content/uploads/2012/05/067-Maher.pdf.

Mateas, M. & Stern, A. (2003). *Façade: An Experiment in Building a Fully-Realized Interactive Drama*. https://users.soe.ucsc.edu/~michaelm/publications/mateas-tidse2003.pdf.

Meehan, J. R. (1977). TALE-SPIN, an Interactive Program That Writes Stories. *Proceedings of the 5th International Joint Conference on Artificial Intelligence, 1*, 91–98. San Francisco: Morgan Kaufmann. https://dl.acm.org/doi/10.5555/1624435.1624452.

Mejía, G., & Niño, K. (2017). A New Hybrid Filtered Beam Search Algorithm for Deadlock-free Scheduling of Flexible Manufacturing Systems using Petri Nets. *Computers & Industrial Engineering*, *108*, 165–176.

Meng, S., Wang, Y., Yang, C-F., Peng, N., & Chang, K-Y. (2024). "LLM-A*: Large Language Model Enhanced Incremental Heuristic Search on Path Planning." https://doi.org/10.48550/arXiv.2407.02511.

Meta Fundamental AI Research Diplomacy Team (2022). Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science, 378*,1067–1074. DOI:10.1126/science.ade9097

Mirzadeh, I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., & Farajtabar, M. (2024). "GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models." https://arxiv.org/abs/2410.05229.

Moore, K. (2026). Human-Analogous Uncertainty Behavior: Cognitive Studies of Large Language Models. Doctoral dissertation, Vanderbilt University, Nashville, TN.

Moore, K., Roberts, J., Pham, T. T., & Fisher, D. (2025). Chain of Thought Still Thinks Fast: APriCoT Helps with Thinking Slow. *Proceedings of the Annual Meeting of the Cognitive Science Society*. https://escholarship.org/uc/item/18x411vv.

Mudgal, S., Lee, J., Ganapathy, H., Li, Y., Wang, T., Huang, Y., Chen, Z., Cheng, H., Collins, M., Strohman, T., Chen, J., Beutel, A., & Beirami, A. (2024). Controlled decoding from language models. *International Conference on Machine Learning, ICML'24*.

Ow, P. S., & Morton, T. E. (1988). Filtered beam search in scheduling. *The International Journal of Production Research*, *26*, 35–62.

Pólya, G. (1954). *Mathematics and Plausible Reasoning*, Princeton: Princeton University Press.

Poole, D. L., & Mackworth, A. K. (2023). *Artificial Intelligence: Foundations of Computational Agents*, 3rd edition. Cambridge University Press.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Riedl, M. O., & Bulitko, V. (2013). Interactive Narrative: An Intelligent Systems Approach. *AI Magazine 34*, 67–67, https://doi.org/10.1609/aimag.v34i1.2449.

Rentschler, M., & Roberts, J. (2026). Reinforcement learning from meta-evaluation: Aligning language models without ground-truth labels. https://arxiv.org/pdf/2601.21268.

Roberts, J. (2024). How powerful are decoder-only transformer neural models?. In *2024 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-8). IEEE. Preprint at https://arxiv.org/abs/2305.17026.

Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., Tay, Y., & Metzler, D. (2022). Confident adaptive language modeling. *Advances in Neural Information Processing Systems, 35*:17456–17472.

Scott, J., & Roberts, J. (2026). "KARMA: Karma-Aligned Reward Model Adaptation." https://arxiv.org/abs/2605.26738.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., & Wu, D. G. (2024). *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*. https://arxiv.org/pdf/2402.03300.

Shin, H., Fisher, D. H., & Anderson, C. B. (2026). AI as Superscholar: Authorship at the Threshold of the Unsayable. *Journal of Theological Librarianship, 19*, 50–65. https://serials.atla.com/theolib/article/download/4538/5933.

Stanley, K. O., Clune, J., Lehman, J., & Miikkulainen, R. (2019). Designing Neural Networks Through Neuroevolution. *Nature Machine Intelligence 1*, 24–35.

Turing, A. (1948). Intelligent Machinery. *The Essential Turing*, edited by B J Copeland, Oxford University Press, 2004. https://doi.org/10.1093/oso/9780198250791.003.0016.

Valiant, L. (2013). *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World*. 1st edition. New York: Basic Books.

Vilalta, R., & Y. Drissi, Y. (2002). A Perspective View and Survey of Meta-Learning. *Artificial Intelligence Review, 18*, 77–95. https://doi.org/10.1023/A:1019956318069.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., & Zhou, D. (2023). Self-Consistency Improves Chain of Thought Reasoning in Language Models. *Proceedings of the Eleventh International Conference on Learning Representations, ICLR '23*.

Wang, X. and Zhou, D. (2024). Chain-of-thought reasoning without prompting. *Advances in Neural Information Processing Systems, 37*:66383–66409.

Wolfe, C. (2025). Group Relative Policy Optimization (GRPO). Retrieved from https://cameronrwolfe.substack.com/p/grpo.

Xie, Y., Kawaguchi, K., Zhao, Y., Zhao, J., Kan, M., He, J., & Xie, M. (2023). Self-Evaluation Guided Beam Search for Reasoning. *Advances in Neural Information Processing Systems, 36*:41618 – 41650. https://neurips.cc/virtual/2023/poster/72456.

Xiong, M., Hu, Z., Lu, X., Li, Y., Fu, J., He, J., & Hooi, B. (2024), "Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs." https://arxiv.org/abs/2306.13063.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of thoughts: deliberate problem solving with large language models. *NIPS '23: Proceedings of the 37th International Conference on Neural Information Processing Systems*, Article No.: 517, 11809 – 11822. Preprint https://arxiv.org/abs/2305.10601

Yao, X., & Liu, Y. (1997). A New Evolutionary System for Evolving Artificial Neural Networks. IEEE *Transactions on Neural Networks, 8*, 694–713. https://doi.org/10.1109/72.572107

YOCHAN Lab, Arizona State University (2026). Retrieved August 20, 2026 from https://yochan-lab.github.io/home/.

Zhao, C., He, S., Jiang, S., Dong, B., Zhao, J., & Liu, K. (2025). Why and How LLMs Benefit from Knowledge Introspection in Commonsense Reasoning. *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 6905–6920, Suzhou, China. https://aclanthology.org/2025.emnlp-main.352/.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., & Wang, Y. (2024). Language Agent Tree Search Unifies Reasoning, Acting, and Planning. *International Conference on Machine Learning*, in *Proceedings of Machine Learning Research 235*:62138-62160. https://proceedings.mlr.press/v235/zhou24r.html.