

Computational Solutions to the Relevance Problem in Cognitive Science: Language Models, Probabilistic Grammars, and Planning Algorithms

Laila Johnston

Preliminary Exam Paper

April 2026

Department of Cognitive and Psychological Sciences, Brown University

Committee: Roman Feiman, Ellie Pavlick, Daphna Buchsbaum, and Christopher Hill

Abstract

Intelligent systems, including human cognition, must be able to identify relevant information in large spaces of possible words, hypotheses, actions, and beliefs. This problem appears across cognitive science under several names, including the frame problem, but it can be stated more generally as the challenge of finding what matters for a task without exhaustively searching the entire problem space. This paper argues that several successful computational frameworks manage this problem through a common strategy: they dynamically reweight their problem spaces so that contextually relevant possibilities become more probable while irrelevant possibilities are pushed to low probability. Importantly, irrelevant possibilities are not eliminated, which allows them to become relevant when the context changes. I examine three case studies from different domains of cognitive science: large language models in language, probabilistic grammars in theory building, and AlphaGo in planning. In each case, I first define the domain-specific form of the relevance problem and then analyze the mechanism by which the system obtains relevant information. Although the underlying mechanisms differ, attention in language models, Bayesian inference with MCMC in probabilistic grammars, and neural-network-guided tree search in AlphaGo, each construct a context-dependent relevant subspace within a much larger space. I conclude by arguing that this insight helps clarify how current computational systems manage relevance while also revealing important open questions about how such relevance weighting is learned.

1. Introduction

Cognitive systems do not only make inferences; in order to make an inference, there must first be a way to determine what elements to make an inference from. For example, if you are looking for a cup in a friend's kitchen, you will need to consider the shape and color of cups and where cups are usually stored in kitchens to succeed in finding a cup. Yet, there is also an infinite amount of other information that is not relevant to finding a cup (such as, the specific wood grain on the cabinets, the brand name on the toaster, a crumb on the floor, and the reflection of the light on the faucet). However, you do not first consider the light on the faucet and then decide it's irrelevant to finding a cup. Your cognitive system somehow knows that the light on the faucet is irrelevant to finding a cup. Further, if the task changes from finding a cup to helping your friend clean their kitchen, some of those previously irrelevant details may suddenly become relevant. Showing the importance of features irrelevant to the past goal (e.g. a crumb on the floor) must not become completely inaccessible, as they are now relevant for a different goal (e.g. cleaning the kitchen). The puzzle is not simply that the world contains too much information. It is that intelligent behavior requires picking out the right subset of possibilities for the current goal without first exhaustively checking everything else.

This challenge appears across cognitive science under several names. In philosophy of artificial intelligence, it is often discussed as the frame problem (McCarthy & Hayes, 1969; Fodor, 1987; Dennett, 1990): how can an agent represent the consequences of an action without also representing the indefinitely many things the action does not affect (e.g., opening the door does not change the color of the sky)? Related challenges include Hamlet's problem, which concerns when to stop deliberating and start acting, highlighting the need to consider relevant options given limited search-time, and the epistemological problem of how an agent ought to update their beliefs after acting in the world in order to reliably track the features that change as a result of the action. In pragmatics, relevance theory (Sperber & Wilson, 2004) approaches a related issue by asking how cognition balances cognitive effect against processing effort. All of these issues are not identical problems and this paper does not treat them as interchangeable. I take these issues to be different instances of a broader computational challenge. I will use the term Relevance Problem to refer to that broader challenge: given an enormous, and in some cases unbounded, space of possible words, hypotheses, actions, beliefs, or percepts, how does a system flexibly identify what matters for the current task without exhaustively searching through everything in the space?

The relevance problem is a problem for any system that behaves intelligently in a large space of possibilities. A language model must generate context-appropriate token sequences from a massive set of tokens and token combinations. A theory-building system must identify plausible hypotheses from a combinatorially large hypothesis space. A planning system must choose useful actions without searching every future branch. These systems were not built to answer philosophers' formulation of the relevance problem. They were built to predict text, construct explanations, and plan moves. However, to succeed at those tasks, these computational

frameworks must implement some way of finding relevant information and making relevant information more likely than irrelevant information.

To gain insights into how to successfully solve the relevance problem, this paper will review computational cognitive science frameworks to investigate the extent to which a computationally instantiable solution to the relevance problem already exists. This paper examines three such case studies from different domains of cognitive science: large language models (LLMs) in language, structured probabilistic grammars in theory building, and AlphaGo in planning. In each case, I first define the domain-specific form of the relevance problem and then analyze the mechanism by which the system obtains relevant information. The aim is not to argue that these frameworks share one mechanism or one unified architecture. The aim is to argue that despite their architectural and computational differences, all three systems manage large problem spaces by dynamically reweighting the space so that a small, context-dependent subset becomes computationally more probable.

An important point of this argument is how relevance is achieved through reweighting strategies. In all three cases, irrelevant possibilities are not entirely not deleted. Instead, irrelevant possibilities are assigned a low probability or low computational priority. This matters because relevance is context dependent. Information that is negligible in one context may become important in another, and a system that permanently removes irrelevant possibilities would lose the flexibility needed to adapt when the task changes. A more plausible strategy, that all three case studies show, is to keep possibilities available while reweighting the space so that, in the current context, some possibilities dominate computation and others don't.

The three case studies instantiate this reweighting strategy differently. In large language models, self-attention and matrix operations reweight the space of possible token continuations relative to the current input. In structured probabilistic grammars, the grammar defines a hypothesis space while Bayesian inference and Markov Chain Monte Carlo (MCMC) focus computational power on searching for hypotheses that better explain the data. In AlphaGo, policy and value neural networks, together with Monte Carlo Tree Search, allow more probable winning game tree branches to be searched instead of searching the full game tree.

One useful way to compare the three case studies is by the different balance they have between constraining a space and searching through a space. Given a problem space, constraining a space means reducing the size of the space to exclude some or all of the irrelevant information given the task. One way to constrain a space is through dimensionality reduction, another is through defining a theory generating grammar. Searching a space means navigating either the full space or the constrained space to find relevant information given the task. Some solutions will rely more heavily on learned constraints, and others on guided search within a reweighted space. I treat this as an interesting point of comparison but not as the paper's primary theoretical claim.

The argument is not to claim that these systems fully solve the relevance problem, nor that they explain how relevance is learned at every level of analysis. There is still very much an open question of how training, learning processes, and initial architectures produce mechanisms

that upweight the right information in the first place. This paper only investigates how already existing systems manage the relevance problem at inference time (i.e. when novel data is shown). Still, investigating how each of these computational systems manage the relevance problem is useful because they show that intelligent performance in large spaces does not require exhaustively checking every possibility, and they offer mechanistic models of how context can make some information computationally salient while leaving other information available for later use.

The rest of the paper proceeds as follows. I begin by situating the relevance problem within a broader intellectual history rather than treating any single definition of relevance as a permanent consensus. I then introduce the comparative framework used throughout the paper: problem spaces, relevant subspaces, reweighting, constraining, search, and context-sensitivity. The three case studies that follow (large language models, structured probabilistic grammars, and AlphaGo) each analyze a different domain-specific version of the relevance problem and the mechanism used to manage it. I conclude by comparing these systems, highlighting both the shared computational pattern and the open questions that remain about how relevance is learned and how these ideas might generalize to human cognition.

2. Background: Historical and Contemporary Formulations of the Relevance Problem

2.1. From the Frame Problem to the Relevance Problem

A good historical starting point for these ideas is the frame problem in philosophy of artificial intelligence. In its original formulation, the frame problem is concerned with how a system could represent the effects of an action without also having to represent the indefinitely many things that the action does not change (McCarthy & Hayes, 1969; Dennett, 1990). If an agent opens a door, for example, it should represent that the door is now open, but it should not also need to derive and represent, one by one, that opening the door did not change the color of the sky, the number of chairs in the room, or the price of tea in China. In this sense, the frame problem is about finding the right computational description of the environment that allows for an agent to represent actions and consequences without generating a combinatorial explosion.

The frame problem is narrower and not identical to the broader issue, the relevance problem, I aim to investigate in this paper. The frame problem expresses the challenge of how an agent represents what changes and what stays the same after an action, but there is the broader issue of how an agent avoids being overwhelmed by the vast amount of information that is available but irrelevant to its current task. For example, suppose an agent is trying to find a cup in a kitchen. A frame-problem-style question asks how the agent should represent the effects of opening a cabinet without also explicitly representing the infinitely many facts that remain unchanged. However, even if that representational issue is handled, a further question remains: out of everything the agent knows about or perceives in the kitchen, which information should it actually use in order to find the cup? The location of cabinets, the typical shape of cups, and the likely places cups are stored may matter. The brand name on the toaster, the pattern in the floor

tiles, or the reflection on the faucet likely do not. The challenge is that the system must identify this relevant subset without first exhaustively checking all the irrelevant information.

Related formulations highlight the same broader pressure in different forms. Fodor's (1987) "Hamlet's problem," for example, concerns when a system should stop deliberating and act, rather than continuing to consider further possibilities indefinitely. An agent can always generate one more inference or consider one more possibility before acting. However, humans must act in a time-sensitive manner. Humans do not sit still, in thought, going through all the implications of what it means to be outside a room containing a ticking bomb (Dennett, 1990). Likewise, problems of belief updating ask which beliefs should become candidates for revision after action or new evidence. These two challenges are not identical to the frame problem, but they share a common structure: a system cannot reconsider everything at once and must somehow figure out what matters for the current task.

This is the broader issue I call the Relevance Problem, the challenge of determining, from a very large space of possible words, hypotheses, actions, beliefs, or percepts, which subset should become computationally prominent for the current task. The frame problem, and related problems, are just some historically important formulations of a broader computational challenge.

2.2. The Frame Problem Debate

The broader issue of the relevance problem started to become more apparent in debates about whether the frame problem was actually an obstacle for artificial intelligence. Fodor (1987) criticized the field, arguing that AI researchers often held contradictory views, simultaneously arguing the frame problem was not worth solving, yet also claiming that AI had already solved it. McDermott (1986) denied that the frame problem existed and thus no solution to it was necessary. He argued that AI systems, in practice, can work by operating in restricted domains, using task-specific representations, and relying on built-in procedures that already narrow what needs to be considered. Once a system is given an appropriate representation of its domain, much of the difficulty of figuring out what is relevant disappears. In other words, McDermott treats domain restrictions and built in architectures as a solution to the frame problem.

Fodor responded by arguing that this attitude is an intellectual blind spot. "If you are unable to recognize the frame problem when as a matter of fact you are having it, you may suppose that you have solved the frame problem when as a matter of fact you are begging it" (1987, pg. 28)). Fodor agreed with McDermott that when information is restricted to what is needed for a specific computation, the system does not need to determine relevance from everything it knows. However, that does not mean the frame problem does not exist. Fodor emphasizes that the difficulty is in global, less bounded areas of cognition, such as planning, reasoning, and inference. In these cases, any belief may be relevant to any other. Including more domain restrictions is not a solution to how the right boundaries are established or shifted when contexts change.

This frame problem debate is important to understand for this paper because it shows that the question to ask is not whether the frame problem exists but how computational systems manage relevance. Current computational systems have already shown reasonable behaviors in very large spaces without leading to combinatorial explosions. But there is still the open question of what computational mechanisms allow successful systems to make some information salient while keeping other information in the background. If domain-specific structures and restrictions are part of the solution, then they should not be treated as dismissals of the relevance problem. Thus, the focus of this paper is not asking whether successful computational systems have solved the frame problem, but how they manage the more general problem of identifying and prioritizing relevant information in large problem spaces.

2.3. Relevance in Contemporary Computational Cognitive Science

Although the frame problem started in classical AI and philosophy of mind, the broader challenge of relevance is not confined to philosophy. In pragmatics, cognitive science, and computational modeling, different frameworks have tried to explain how cognitive systems avoid being overwhelmed by the amount of information they could in principle process. These frameworks do not all use the term *relevance problem*, and they do not all operate at the same level of analysis. But, what they show is that intelligent behavior depends on some mechanism for making certain information matter more than other information in a given context.

Let's start with research from the pragmatics literature. In Relevance Theory, Sperber and Wilson (2004) argue that cognition is geared toward maximizing relevance, where relevance reflects a tradeoff between cognitive effect (the value of the new information) and processing effort (the cost of deriving that information). This framework is important for this paper because it captures something that is important to relevance: what matters is not simply whether information is true or available, but whether that information is worth processing for the task at hand. Probabilistic models of pragmatics, such as Rational Speech Act models, have also used this idea by modeling communication as maximally informative inference over contextually relevant alternatives (Goodman & Frank, 2016). Even though these approaches don't address all aspects of relevance (for example, where do contextually relevant alternatives come from), they show that relevance can be modeled.

Another attempt at figuring out what the mechanisms of relevance are comes from the predictive processing literature. The predictive processing view holds that cognition depends on a form of hierarchical prediction and prediction error minimization (Friston, 2010; Clark, 2013). Not all prediction errors are treated equally, and not all features of the environment receive the same compositional priority. Recent work has connected this view to the relevance problem, arguing that predictive processing provides a way of addressing the relevance problem through the prioritization of signals relative to context and task demands (Darling, Corcoran, & Hohwy, 2025). I do not argue that the case studies examined later are reducible to predictive processing, but this literature is important because it views relevance as a problem of selective weighting rather than exhaustive search.

A third line of work addresses relevance through resource-rationality and approximate inference. Resource-rational approaches treat cognition as an efficient use of limited computational resources rather than as an ideal inference unconstrained by time or memory (Griffiths, Lieder, & Goodman, 2015; Lieder & Griffiths, 2020). A sampling-based view of cognition argues that intelligent behavior often depends on approximate rather than exhaustive computation. Rather than exhaustively searching a problem space, sampling algorithms instead spend computational effort and time on a small local region on the problem space (Sanborn & Chater, 2016; Sanborn & Chater, 2017; Zhu et al., 2024; Castillo et al., 2024). This literature shows that tractable cognition often requires not considering everything equally. From this perspective, relevance is closely tied to how computation is distributed under constraints. It also connects naturally to Fodor’s Hamlet’s problem: if a system cannot evaluate every possibility, it must have some way of allocating limited computation and of stopping deliberation in time to act. Resource-rational and sampling-based approaches provide some solutions to the boundedness of cognition and computation.

A final important line of research comes from amortized inference and structured learning. Rather than treating each new problem as if inference begins from scratch, amortized approaches seek to understand how previous computation can be reused so that later inference becomes faster and more selective (Gershman & Goodman, 2014; Dasgupta et al., 2018; Dasgupta et al., 2020). Related work on abstraction, structured concept learning, and probabilistic program induction show that large hypothesis spaces become tractable when they are organized by reusable structure and compositional priors (Tenenbaum et al., 2011; Lake, Salakhutdinov, & Tenenbaum, 2015; Lake et al., 2017). These ideas will be relevant for the discussion of probabilistic grammars, where the structure of the hypothesis space is itself part of what makes selective relevant inference possible. There is also a connection to the epistemological problem, since updating beliefs after action or new evidence cannot require reconsidering every stored belief from scratch. Reusable structure and amortized inference help explain how some hypotheses or belief updates become candidates for further computation while others remain in the background.

All these literatures show that many researchers are attempting to explain selective processing, bounded search, and context-sensitive prioritization. So, it is not the case that cognitive science has no mechanistic ideas of how relevance works. The gap is that there is no settled account of relevance that spans this literature. As we can see, different frameworks emphasize different mechanisms: utility trade-offs in pragmatic reasoning, prediction weighting, efficient allocation of limited resources, approximate sampling, reusable inference, and structured abstraction. The aim of this paper is not to argue for a single theory of relevance. Instead, it is to evaluate the three case studies that follow within this broader landscape and to ask what can be learned by comparing successful computational systems that manage relevance in practice.

2.4. What relevance is in this paper

In this paper, the Relevance Problem refers to the general challenge of, given a very large space of possible words, hypotheses, actions, beliefs, or percepts, how does a system choose the right subset of possibilities for the current task without exhaustively evaluating everything else? To say that some information is relevant is not necessarily to say that it is true, rational, or ultimately correct. It is to say that the information is worth prioritizing for the system's present goal. A successful system must somehow bring relevant possibilities to the foreground of computation while leaving others in the background. For clarification, the problem spaces discussed in this paper are not always literally infinite. Most of the time the problem spaces are finite but so large that exhaustive search is computationally intractable. What matters is how the conflict between the size of the space and the limited time, memory, and computation that is available to the system is handled.

A very important point to emphasize is that relevance is context dependent. The same information may be negligible in one setting and important in another. A crumb on the floor may be irrelevant if the task is to find a cup, but relevant if the task is to clean the kitchen. Because of this, a solution to the relevance problem cannot be to just exclude irrelevant information. The challenge is to have a system that can shift what information is treated as relevant when the task, evidence, goal, or environment changes. The case studies in this paper were partly chosen because each of them succeed at prioritizing context-dependent information within a much larger space of possibilities.

Intentionally, this way of defining relevance is broad enough to cover many domains. In language, the challenge is to prioritize token continuations or interpretations over many alternatives. In theory building, it is to focus computation on plausible hypotheses rather than the full hypothesis space. In strategic planning, it is to focus computational resources to winning actions and futures rather than exhaustively expanding all available branches. These domains differ in architecture and purpose, but they share the same general computational pressure: intelligent behavior requires selecting what matters without considering everything equally. The aim of this paper is not to show that all successful systems share one identical algorithmic mechanism, nor to claim that any of them fully resolves the relevance problem. The aim is to compare how different systems manage this pressure in practice.

3. The Comparative Framework

In Section 2, I argued that the relevance problem is a broad computational pressure that appears across many frameworks in cognitive science. In this section, I will introduce the vocabulary I will use to compare the three case studies. My goal is not to claim that large language models, probabilistic grammars, and planning systems share one identical algorithmic mechanism to manage relevance. The aim is to identify a common computational pattern to managing relevance (what I call dynamic reweighting) that appears across these domains despite architectural differences.

3.1. Problem Spaces and Relevant Subspaces

For any cognitive task, a system operates with a problem space, such as a set of possible words, hypotheses, actions, beliefs, or percepts that could enter into computation. The size and structure of this space varies across domains. In language, it may involve possible token continuations or utterance interpretations. In theory building, it may involve candidate explanatory hypotheses. In planning, it may involve available actions and future trajectories. In each case, the problem space is too large for every possibility to get equal computational attention.

Thus, for a system to succeed on some cognitive task, the system needs to form some smaller relevant space from that larger space. This is what I will call a relevant subspace. A relevant subspace is not the set of all true or correct possibilities or the final solution to the task at hand, it is a subset of the possibilities that the system prioritizes in computation given its current goal. The relevant subspace is what is made salient to guide inference, interpretation, or action.

I intentionally want these terms to be broad enough to cover many different implementations. Different systems may construct relevant subspaces in different ways. As we will see, some do so through learned statistical weighting, others through structured priors, and others through a combination of learned evaluation and explicit search. Differences in these algorithmic instantiations is of course interesting, but my goal here is to argue for a more general principle of how the relevance problem is managed in current systems.

3.2. Dynamic Reweighting

The central claim of this paper is that the three case studies all manage relevance by dynamically reweighting their problem spaces. By reweighting, I mean that a system can change the computational priority of the elements in the space so that some elements become more important while other elements become less important. This does not mean permanently deleting low importance or irrelevant elements. Instead, irrelevant elements are kept but assigned low probability, low value, or low priority relative to the current context.

Not deleting irrelevant elements matters because relevance is context sensitive. Information that is negligible in one context may become needed in another context. A system that erased low-priority possibilities would lose flexibility when the task or evidence changes. A more successful strategy that the case studies point to, is to keep all elements of the problem space but shift their relative weights. Here, relevance is managed by reorganizing the space so that a context-dependent subset becomes computationally dominant at a given moment.

This principle of dynamic reweighting is not a theory of learning or cognition. It is just a computational principle, that I identified, that the three case studies share even though they differ in architecture and purpose. My explanation of dynamic reweighting does not explain how a system comes to learn the right weighting function in the first place. That is a current limit on the argument.

3.3. Space Constraining and Space Searching

Managing relevance through dynamic reweighting is the central claim of this paper. However, through investigating the case studies, another point of comparison was identified. The three case studies differ in how they balance constraining a space and searching through a space when it comes to finding a relevant subspace. These terms are not mutually exclusive, some systems can use a mixture of constraining and searching.

Constraining refers to a process that reduces the size of the larger problem space. This can mean that a system puts more computational resources on a smaller portion of the problem space (instead of using the same amount of resources on every part of the problem space). The effect of constraining is that computation is concentrated on a smaller region of the problem space. Searching a space refers to a process for navigating the problem space in order to identify relevant elements. Searching can happen in the full problem space or in a subspace that has already been constrained.

There is a blurry line between constraining and searching because these two terms are closely related. A search algorithm may depend on prior constraining, and a constraining mechanism can guide where the search should happen next. Is a really efficient search algorithm good enough (no space constraining needed)? Or is space constraining all that is needed (the search doesn't matter)? Or are both needed to determine relevance? In practice, it is often not the case that a system relies solely on constraining or solely on searching. Usually the two work together. However, the reason to distinguish between these two notions is because it provides another dimension of comparison. Now, we can investigate whether a system relies more heavily on narrowing the space in advance, on actively exploring within that space, or on a combination of both.

This distinction is also useful for the case studies that follow. Large language models rely heavily on reweighting that makes a small set of continuations computationally dominant (reducing the vocabulary to a few likely words). Probabilistic grammars lean on structured search procedures that concentrate on promising hypotheses (navigating a grammar). AlphaGo has the most explicit hybrid approach, using neural networks to constrain the parameter of the game tree so that algorithms can efficiently search it. These systems differ in how much they depend on constraining and searching, but all three must still solve the broader problem of making some possibilities matter more than others.

3.4. Plan for the Case Studies

In the next sections, I will apply this framework to three domains: language, theory building, and planning. In each case study, I ask the same questions. What is the problem space? By what mechanism is the space dynamically reweighted? What becomes relevant, and what recedes into the background? What different roles do constraining and searching play? My goal is to identify and clarify a recurring computational strategy to manage relevance across these three domains. In each case, intelligent performance in large problem spaces depends on dynamically producing a tractable, context-dependent subset of possibilities.

4. Case Study 1: Large Language Models

4.1. The Relevance Problem in Language

Language is a very clear instance of the relevance problem. In conversations, both the speaker and listener operate in a space full of possible next words, interpretations, references, discourse moves, and questions under discussion. A speaker must generate an utterance from a vast vocabulary space that is appropriate to the current discourse context and best expresses the meaning they want to share. While a listener must infer what the speaker likely meant given both what they actually said and the alternative utterances they could have said. If the cognitive system of the speaker and listener had to actively check every possible word or alternative against the current context to verify its relevance, the conversation would never continue. Thus, language communication depends on treating some possibilities as immediately worth considering and others as negligible.

Further, both the production and comprehension of language can also happen in an ad-hoc way (i.e. the relevant possibilities are not fixed in advance). For example, the word “bank” does not produce the same utterance continuation or interpretation in a conversation about rivers as in a conversation about finance. What is relevant is dynamic and depends on the context and the broader communicative intent. This point becomes more clear when human language communication is compared with language models that rely on non-contextual lexical representations. In a static embedding model, “bank” is assigned the same vector regardless of whether the context concerns finance or rivers. This vector will usually capture the aggregate statistical association of “bank” across many uses (including the finance use and river use). In this case, the representation of “bank” is not formed by making one use of “bank” computationally prominent and another use negligible as context changes. So, it is not simply about whether a system stores information of words, but whether it can dynamically prioritize the interpretation or continuation that is relevant for the current context.

Large language models are useful for the present paper because they make the relevance problem in language computationally explicit. The general ability of modern LLMs to engage in coherent and context-dependent conversations is evidence that a computational architecture can, at least seemingly, determine what is relevant from a large space. A transformer language model is trained to predict the next token given a preceding context (Vaswani et al., 2017). However, performing this task well requires modeling context dependencies across tokens, not just relying on fixed associations between words considered in isolation (Brown et al., 2020). In other words, LLMs must assign high weight to a small set of token continuations that fit the current input text while assigning low weight to the majority of token continuations that are grammatically possible or lexically available but contextually inappropriate. In this sense, language modeling provides a computational setting in which the relevance problem can be studied mechanistically.

The success of LLMs highlights another important point, namely that LLMs were not designed to solve the philosophical formulation of the frame problem. They were designed to perform next-token prediction. Yet success on next-token prediction depends on some way of

making contextually relevant tokens computationally prominent and irrelevant tokens computationally negligible. A language model that could not do this would not be able to sustain coherent discourse, track reference across sentences, or adjust its outputs as the topic shifts.

The relevance problem in this domain can be stated as, given a large space of possible token continuations and interpretations, how does a model make a small, context-appropriate subset computationally dominant for any kind of input text? The claim is that large language models manage the relevance problem by dynamically reweighting the space of possible token continuations relative to the current input. The next subsection is about the mechanism behind that claim. The argument is that attention-based mechanisms allow LLMs to weight information from the input sequence differently, producing a distribution in which some continuations become highly salient and others recede into the background. This does not fully explain how such relevance is learned during training, but it does show how relevance is managed at inference time.

4.2. Attention as a Mechanism for Dynamic Reweighting

In this section, I argue that the core mechanism that allows a transformer language model to manage relevance at inference time is self-attention. Unlike language models that rely on static lexical representations, where each token has fixed representation regardless of context, in a transformer the representation of each token is repeatedly recomputed as a function of other tokens in the sequence. This repeated computation over the tokens in the sequence is what allows for the dynamic reweighting of token representations.

Before continuing, a note on what “context” means in this setting. Loosely, context means the prior input text the language model has available for the current prediction. More specifically, the context for predicting the next token is the sequence of prior tokens within the model’s context window. As a helpful example, if the context window is 5 words, then for the sentence, “All models are wrong but some are”, the LLM would only use “are wrong but some are” to predict the next word. In practice, modern LLMs often have very large context windows, so this paper does not treat context window size itself as the central issue. The more important point is that, even within a context window, the model must still determine which prior information should matter more for the current prediction.

Earlier language models built from different architectures, such as recurrent neural networks, are also context-sensitive in the sense that the representation for each new predicted token depends on the tokens that came before it in the sequence. In recurrent neural networks, the information from every token in the sequence gets compressed and carried forward to the next token in the sequence through the hidden states. Thus, by the end of the sequence, the context information is a compressed average. This means that the influence of prior tokens is indirect, and information from distant parts of the sequence can be harder to preserve or selectively recover. The advantage of the transformer is that the contextual weighting over tokens is more explicit and flexible. In a transformer, each token position can assign different weights to earlier positions in the sequence rather than relying only on a single compressed summary passed

forward step by step. The model does not only inherit a general summary of the past, but can selectively increase or decrease the influence of particular earlier tokens depending on the current prediction.

Formally, in a transformer layer, each token representation has three vectors, a query (q), a key (k), and a value (v). If the current token representation at position i is x_i , the model computes

$$q_i = x_i W_Q, \quad k_i = x_i W_K, \quad v_i = x_i W_V$$

where W_Q , W_K , W_V are the learned parameter matrices. These vectors are not goals, intentions, or explicit symbolic descriptions. They are learned linear projections used to determine how strongly different positions in the sequence should influence one another. Every token in the context is projected into a key vector, and the reweighting happens when the model compares the current query to all of those keys.

For the current token position i , the model compares the current query q_i with the keys at all the other positions. Using a scaled dot-product, the model computes a compatibility score between the current token and each prior token in the context. These scores are then passed through a softmax, which converts them into attention weights. The weights sum to one and determine how much each prior token contributes to the updated representation at the current token position. The model then forms the updated representation as a weighted combination of the value vectors from the context.

This is the sense in which attention implements dynamic reweighting. All prior tokens within the context window remain available, but they do not contribute equally to the computation. Tokens whose keys are more compatible with the current query receive higher weights and therefore exert greater influence on the updated representation. Tokens that are less useful in the current context receive lower weights and exert less influence. Importantly, low-weight information is not deleted from the model's representational space. It is still available, but it matters less in the current prediction. This attention mechanism allows the model to suppress some contextual information without losing the possibility that the same information could become important in a different context.

Let's look at an example. Consider the token "bank" in a context about rivers and in a context about money. In a transformer, the representation computed for "bank" will not be the same in both cases, because the surrounding tokens will change the weights used to construct this specific representation of "bank". In a river context, tokens associated with water, location, or fishing may receive higher weights. But in a financial context, tokens associated with transactions, cash, or institutions will instead receive higher weights. The point is that the context changes which tokens become more influential for the next stage of computation.

Next, it is important to distinguish the training objective from the inference-time mechanism. During training, the model predicts the next token in many different text sequences, and its parameters are adjusted through backpropagation so that its predictions become more accurate over time. This training process shapes the learned query, key, and value matrices. At

inference time, however, the model is no longer updating its parameters. Instead, it uses those learned matrices to compute attention weights over the current context. In other words, next-token prediction during training helps create the matrices, while attention at inference time is the online mechanism that actually reweights contextual information for a particular prediction.

Finally, attention does not directly reweight the vocabulary itself. It reweights the contextual information used to build hidden representations, and those hidden representations are then transformed into a probability distribution over possible next tokens. In this case study, the relevance claim can be stated as, attention is a mechanism for dynamically reweighting the context information, and this helps output a distribution in which some token continuations become much more probable than others.

4.3. Constructing Relevant Subspaces in Transformers

The previous subsection argued that attention is the mechanism by which a transformer dynamically reweights contextual information. That mechanism matters because it helps explain how a large language model constructs a relevant subspace within a much larger space of possible continuations. Note that attention itself does not choose the next token. By reweighting contextual information, attention produces hidden representations that are then transformed into a probability distribution over the model’s vocabulary.

In the final steps of the token prediction process, the model maps its final hidden representation into a set of scores over possible next tokens and applies a softmax function to convert those scores into probabilities. The result is a distribution over the entire vocabulary. This distribution is often highly uneven: a relatively small set of continuations receives most of the probability mass, while a much larger number of tokens are assigned very low probability. Because the model has already made some prior information more influential than others, only a limited portion of the vocabulary emerges as strongly compatible with the current input.

This is the sense in which the model constructs a relevant subspace. The full problem space is the large set of possible next tokens the model could produce. The relevant subspace is the small portion of that space that becomes computationally dominant for the current prediction. This subspace is not fixed in advance. It changes as the context changes. A continuation that receives low probability in one context may receive high probability in another, not because the token has been deleted and reintroduced, but because the contextual reweighting has changed which continuations are most compatible with the input.

It is useful to distinguish this relevant next token distribution from the later step of decoding. After the model has produced a probability distribution over next tokens, a decoding strategy such as greedy decoding, beam search, or sampling determines how one token is actually selected. These decoding procedures matter for the final output, but the core relevance work already occurred when the model constructed a distribution in which some continuations became much more prominent than others. Token decoding operates over the already contextual reweighted space.

Understanding relevance through probability reweighting can be useful. If irrelevant possibilities were completely removed, the model would lose flexibility when context changes. Instead, low-probability possibilities remain available. Their contribution is suppressed, not erased. This allows the model to adapt when a different prompt makes those possibilities newly relevant. In that sense, the model manages relevance by reorganizing the space of possible continuations rather than by permanently eliminating most of it.

However, relevance through dynamic probability reweighting also has limitations. A low-probability token may not be the same thing as an irrelevant token in a philosophical sense. Probability is the model’s internal measure of compatibility with the current context, not a direct label of relevance. Still the output next-token distribution gives a concrete computational indicator of which possibilities the system is treating as worth prioritizing.

This is why large language models are a useful case study for the broader argument. They show, in a mechanistically explicit way, how a system can manage a very large problem space without exhaustively evaluating all possibilities equally. Attention reweights contextual information, the resulting hidden representations generate an uneven output distribution, and decoding operates over that already structured space. In the language domain, that is how the current computational systems construct a relevant subspace.

5. Case Study 2: Structured Probabilistic Grammars and Theory Building

5.1. The Relevance Problem in Theory Building

The second case study concerns theory building rather than language prediction (i.e. selecting the next word in a sequence). In this domain, the problem is to infer an explanatory hypothesis from limited data. A learner observing a novel pattern of events, for example, objects interacting on a table, may need to infer hidden causal structure such as force, mass, collision, or magnetism in order to explain what was observed. The challenge is that the space of possible explanations is extremely large. For almost any dataset, many different hypotheses could be constructed, and those hypotheses may vary widely in complexity, plausibility, and explanatory value. Further, new data can change hypotheses, confirm hypotheses, or create entirely new hypotheses. If all candidate explanations had to be considered equally, theory construction would quickly become intractable.

In this domain, the relevance problem is at the level of hypothesis generation. In language modeling, the system must identify relevant continuations from a large space of possible outputs. In theory building, the system must identify relevant explanations from a large space of possible hypotheses. The relevance problem here can therefore be stated as, given limited data and a very large hypothesis space, how does a system concentrate computation on the small subset of hypotheses worth considering?

The framework examined in this case study comes from work on structured probabilistic grammars and theory learning, especially Ullman, Goodman, and Tenenbaum (2012). More

broadly, this work belongs to a larger line of research that treats learning as inference over structured, compositional hypothesis spaces rather than as association alone (Tenenbaum et al., 2011; Lake et al., 2015; Lake et al., 2017). This literature emphasizes that hypothesis spaces are not just large, they are also structured. What makes theory learning possible is not only that some hypotheses are more probable than others, but that the space itself is organized by reusable primitives and compositional rules. For that reason, the probabilistic grammar case is especially useful here because it shows that the relevance problem arises not only in choosing among possibilities, but in generating and organizing a space of possible explanations.

In this case study, the problem space is the space of candidate theories or hypotheses that could explain the data. The relevant subspace is the smaller set of hypotheses that become worth pursuing given the observed evidence and the structure of the learner’s representational resources. A relevant hypothesis here does not simply mean a strictly true hypothesis. A hypothesis can be relevant in the computational sense that it is worth evaluating, comparing, or refining further, even if it ultimately turns out to be wrong. The relevance problem in theory building is therefore both a matter of finding the correct theory to explain the data and of avoiding blind search through a vast space of possible hypotheses.

The claim of this section is that structured probabilistic grammars manage the relevance problem through a combination of structured generation and probabilistic reweighting. The grammar organizes what kinds of hypotheses can be constructed, while Bayesian inference and stochastic search concentrate computation on hypotheses that better explain the observed data. The system does not begin by evaluating every possible theory equally. Instead, it constructs and reweights a hypothesis space so that a comparatively small subset becomes computationally prominent for the current data. The next subsections explain how the grammar organizes a hypothesis space and the role of Bayesian inference and MCMC in navigating that space.

5.2. The Grammar as Structuring the Hypothesis Space

The first component of the theory building through structured probabilistic grammars framework is the probabilistic grammar. The grammar provides the representational resources from which candidate theories can be constructed. The grammar itself does not directly select the most relevant or correct theory from all possible theories. Thus, the grammar does not by itself solve the relevance problem. What it does is shape the larger problem space by determining what kinds of hypotheses are expressible in the first place.

The role of the grammar is important because theory building requires more than choosing among a fixed or known list of options. A learner encountering new data must be able to construct candidate explanations that fit the observed pattern. In structured probabilistic grammar models, this is achieved through a set of primitives and combinatorial rules that generate possible hypotheses compositionally. The resulting space is large, but structured. Some kinds of explanations can be built, others cannot, and the complexity of hypotheses depends on how those primitives are combined.

One way this theory building framework manages relevance is through the structured hypothesis spaces. If the learner had to search over a completely unorganized space of arbitrary candidate theories, inference would quickly become intractable. The grammar imposes a prior organization on the space. It defines the dimensions along which hypotheses can vary and gives the learner a way to generate candidate explanations that are appropriate to the general kind of problem being faced. In that sense, the grammar helps make the problem space manageable before any data is taken into account.

Using the example and empirical experiments from Ullman, Goodman, and Tenenbaum; 2012, suppose a learner observes a pattern of object movements and must explain why one block moves when another approaches it. A grammar with primitives for objects, forces, collisions, and attraction can generate hypotheses that explain those object observations in terms of the relations between the primitives. This is already a significant restriction on the larger space of imaginable explanations. But it still does not tell the learner whether the best explanation in this particular case involves collision, magnet-like attraction, or some more complex combination of causal rules. The grammar makes such hypotheses available but it does not decide among them.

The contribution of the grammar comes with an important limitation. The grammar places strict bounds on what the system can represent in the first place. Any observed data must be describable in terms of the grammar's available primitives, and those primitives are typically selected in advance by the researchers on the basis of their assumptions about the relevant structure of the domain. For example, in the model of object interactions (from Ullman, Goodman, and Tenenbaum; 2012) the primitive vocabulary includes notions such as object, force, motion, collision, and attraction. This helps make the hypothesis space tractable, but it also means that the system can only generate hypotheses that are expressible in that formal language. If the data observed is not object based, such as observing two people holding hands and inferring they are romantic partners, the grammar has no way to represent this data. Further, even if the data is representable by the grammar, if the relevant causal mechanisms for explaining new data lie outside the grammar's primitives, the model will fail to find the relevant theory. For example, if the blocks are moving due to magnetism, but the grammar only contains primitives for collisions, the model cannot generate the relevant hypothesis. It will attempt to explain the data using only collisions, failing to find a theory that best explains the data.

Thus, the grammar does not provide an unconstrained solution to the relevance problem. It helps manage relevance by imposing an organized representational structure in advance, but that very organization depends on prior assumptions about which features of the environment are worth representing. The learner is not constructing theories from a completely open-ended space. The learner is constructing theories from a space that has already been shaped by a particular formal language.

To summarize the contribution of the grammar to managing relevance, the grammar determines the form of the theory space within which relevant theories will be assessed, but does not itself directly identify the relevant theory. The grammar helps organize the overall problem space so that the system is not searching over an unstructured infinity of possibilities. The next

subsection explains how Bayesian inference and MCMC reweight that structured space so that a smaller subset of hypotheses emerge as relevant for the current data.

5.3. Bayesian Reweighting and Stochastic Search

If the grammar structures the hypothesis space, the next question is how the system identifies which hypotheses are worth pursuing for a particular dataset. This is where Bayesian inference and stochastic search come in. The grammar makes candidate theories expressible, but it does not determine which of those theories best explain the evidence. The role of Bayesian inference is to reweight the structured hypothesis space relative to the observed data, so that some theories become more plausible than others. The role of MCMC is then to search through that reweighted space without exhaustively evaluating every possibility.

At the Bayesian level, a candidate theory is evaluated in terms of its posterior probability given the data. This posterior depends on two factors: a prior, which reflects how probable the theory is independently of the current data, and a likelihood, which reflects how well the theory explains the observed evidence. In the Ullman et al. (2012) framework, the prior is partly computed from the grammar, since the grammar makes some theories easier to generate than others. This means that the system does not begin from a blank space of equally weighted theories. Simpler or more readily generated theories already have an advantage, while more complex theories are favored only if they explain the data better. Bayesian inference is the mechanism that reweights the hypothesis space so that some regions become more relevant than others.

In the full theory space, there may be indefinitely many candidate explanations. But after the prior and the observed data are taken into account, only a much smaller region of that space will have substantial posterior support. These are the hypotheses that become computationally salient. This is what we can call the relevance subspace. Importantly, this does not mean that a learner building a theory for some observed data explicitly computes and stores the posterior probability of every theory in the space, and then picks a theory from that list. That would be impossible in a huge or infinite space. However, the posterior gives an ideal description of which theories should count as more plausible than others. The algorithmic question is then, how can a learner explore that space in practice without exhaustively evaluating every candidate theory?

One solution is a Markov Chain Monte Carlo (MCMC) sampling algorithm. In the Ullman et al. (2012) model, the learner does not begin by computing the probability of every possible theory and then sampling from that complete list. Instead, MCMC starts from a random current theory in the space and proposes a local modification to that theory. A proposal might add a rule, delete a rule, replace one predicate with another, or resample part of the theory from a rule template. The new proposal is then compared to the current theory, and the learner probabilistically decides whether to accept the move. Thus, MCMC explores the theory space step by step as a local stochastic walk, rather than by first evaluating the whole space.

MCMC does not require computing $P(H|D)$ for every possible hypothesis H in an infinite space before search can begin. What MCMC does at each step is a comparison between

the current theory and a newly proposed one. In a Metropolis-Hastings procedure, the decision to accept a move depends on a ratio involving the posterior score of the proposed theory, the posterior score of the current theory, and the proposal probabilities for moving between them. The algorithm therefore works by making many local comparisons, not by globally evaluating the entire hypothesis space in advance. This is why MCMC can be used in spaces that are too large for exhaustive Bayesian calculation.

However, the MCMC mechanism is not super straightforward because local comparisons between two theories require some way of estimating how well each theory explains the observed data. This is nontrivial in the Ullman et al. (2012) framework because a theory often contains hidden predicates or abstract concepts that are not directly specified in the observations. In other words, to evaluate a theory, the learner must not only ask whether the theory is plausible in general, but also how that theory should be interpreted given the particular dataset.

Suppose a learner observes three objects, A, B, and C, and notices that A attracts B, while A does not attract C and B does not attract C. Now consider a candidate theory that says, two objects interact if both are magnetic and at least one of them is a magnet. Even if this theory is probable, the learner still needs to determine how well it explains the data. The theory contains hidden predicates, such as being magnetic or being a magnet, and these are not directly given in the observations. To evaluate the theory, the learner must also consider which assignments of these hidden properties to A, B, and C would make the observed interaction pattern most likely. For instance, perhaps A and B are both magnetic and A is a magnet, while C is not magnetic. But other assignments are also possible, and some will fit the data better than others.

Thus, Ullman et al. (2012) model theory learning as a nested search process. The outer loop searches over theories by proposing changes to the rules themselves. The inner loop searches over ways of instantiating a given theory on the observed data by assigning values to its hidden predicates. In the example above, the outer loop might compare a collision-based theory with a magnetism-based theory, while the inner loop asks, for the magnetism theory, which specific objects should count as magnetic or as magnets. The overall process is therefore both a search over candidate theories and a search over how those theories apply to particular cases.

This nested structure helps clarify the relation between Bayesian reweighting and stochastic search. The posterior provides the normative landscape by showing which regions of theory space should count as more plausible given the prior and the data. MCMC then provides a local algorithm for exploring that landscape. The learner does not begin with a complete list of pre-scored hypotheses, because even scoring a single candidate theory requires further inference about how its hidden structure maps onto the observed data. Instead, the learner uses stochastic search at two levels, an outer loop over candidate theories and an inner loop over the possible interpretations of those theories. Here we can say that Bayesian inference reweights the space and MCMC concentrates search within that reweighted space.

In many Bayesian models, MCMC is simply a computational device used by the modeler to approximate an otherwise intractable posterior. Ullman et al., (2012) however, claim something stronger. They propose that a stochastic search process may also capture something

about the learner’s own theory-construction dynamics. This paper is agnostic to that stronger interpretation but it means that this kind of search process has been proposed as an algorithmic account of how a learner could manage relevance in theory learning.

To summarize, relevance in theory building is managed through a layered process. The grammar structures what can be represented, Bayesian inference reweights the space of candidate theories in light of the observed data, and MCMC concentrates search in regions of that space with higher posterior support. The result is that a relatively small subset of candidate explanations becomes computationally prominent, even though the larger hypothesis space remains in principle available for further exploration.

6. Case Study 3: AlphaGo and the Relevance Problem in Planning

6.1. The Relevance Problem in Planning

The third case study examines strategic planning, specifically game planning in the domain of Go. In this setting, the relevance problem does not arise as a selection of next words (as in case study 1) or the construction of an explanatory hypothesis (as in case study 2), but as the problem of deciding which actions and future trajectories deserve computational attention given the goal. At any given board position in Go, a player faces many legal moves, and each move opens into further branches of possible play. The difficulty is not simply that there are many possibilities at one step. It is that each move leads to an expanding tree of future positions, making exhaustive search computationally infeasible. In planning, then, the relevance problem can be stated as, given a large space of possible actions and future trajectories, how does a system identify the comparatively small subset worth exploring further?

Go makes this problem especially noticeable. The number of possible board states is finite, but astronomically large (approximately 10^{170}), and the branching factor at each turn is much higher than in a game like chess (~ 250 moves per turn in Go compared to ~ 35 in chess). Both games require planning over large trees of possible moves, but Go presents a much larger branching factor and a search space that is more difficult to explore exhaustively. A planning system that treated every legal move and every possible continuation as equally worthy of attention would quickly become overwhelmed. As a result, the central challenge in Go is not only to efficiently search, but to determine which small subset of moves and futures is worth searching in the first place.

A similar point can be made from the perspective of human expertise. Strong Go play does not appear to depend on treating every legal move as equally worthy of deliberate evaluation. Rather, expert performance depends on rapidly concentrating attention on a relatively small set of promising moves (Nakatani & Yamaguchi, 2014). This again highlights that the relevance problem in planning requires not just searching well, but determining where search is worth concentrating in the first place.

For these reasons, Go became such an important test case for artificial intelligence. Earlier game-playing systems, especially classical chess engines such as Deep Blue, combined

search with hand-crafted evaluation functions, move ordering, and pruning heuristics (Campbell, Hoane, & Hsu, 2002). The successful winning Go algorithm, AlphaGo, did not introduce selective search for the first time. However, AlphaGo required new algorithmic innovations because Go’s search space is enormous and board positions are difficult to evaluate using relatively fixed, hand-engineered heuristics alone (Silver et al., 2016). In this domain, successful planning requires dynamically prioritizing promising moves, estimating the value of resulting positions, and using those signals to guide how search proceeds through the game tree.

AlphaGo offers a concrete computational example of how relevance can be managed in a planning problem with the kinds of challenges Go faces. Its importance is how its components (stochastic tree search and neural networks) work together to make some moves and futures computationally prominent and others comparatively negligible. Here, the full problem space is the large space of legal moves and possible game trajectories, while the relevant subspace is the smaller region of that space to which the system devotes most of its computation in a given board state.

The claim of this section is that AlphaGo manages relevance through a combination of learned evaluation and guided search. Its policy and value networks help reweight the planning space by making some moves and board states more promising than others, while Monte Carlo Tree Search uses that information to concentrate exploration on a smaller portion of the game tree. The next subsections explain these components and show how, together, they allow the system to avoid treating every possible continuation as equally important.

6.2. The Policy Network as Reweighting the Move Space

The first learned component in AlphaGo is the policy network, which evaluates legal moves relative to a given board position. The policy network does not predict the ultimate winner of the game or perform search in the game tree. Rather, it provides a probability distribution over possible next moves. In the terms of this paper, the policy network helps construct a relevant subspace of actions by making some moves more probable than others in the current board state.

As mentioned in section 6.1, the planning problem in Go is that at any given position, there are a large number of legal next moves. The system cannot treat all of them as equally important, because each move opens onto further branches of future play. What is needed is some way of distinguishing moves that are worth further exploration from those that are unlikely to be useful. The policy network outputs exactly this kind of information. Given the current board state as input, it outputs a distribution over legal actions, assigning higher probability to moves that it has learned to regard as relevant in that context and lower probability to moves that appear less useful to winning the game.

As an example, suppose AlphaGo is evaluating a board position in which many moves are available. The policy network may assign relatively high probability to a small number of strategically promising actions, such as defending a weak group, extending influence into an important region, or responding to an immediate local threat, while assigning much lower

probability to moves that are legal but strategically unpromising in that same position (probability of 0.40 to Move A, 0.35 to Move B, and <0.001 to the remaining 248 moves). The important point is not that low-probability moves become impossible. It is that the policy network makes a relatively small subset of actions stand out as much more worthy of further search.

In this way, the policy network can be understood as a mechanism of dynamic reweighting. The same move may be assigned a very different weight depending on the surrounding board state. A move that is highly probable in one position may be assigned negligible probability in another. Relevance in this case is therefore not an intrinsic property of the move itself. It is a relation between the move and the current state of the board. The policy network operationalizes that relation by producing a context-sensitive distribution over available actions.

This is one thing that is distinctive about AlphaGo relative to earlier game-playing systems. Classical game engines also used evaluation and move-ordering heuristics, but these were generally hand-engineered and relatively fixed in advance. AlphaGo's policy network instead learns, from data, a context-sensitive mapping from board states to move probabilities. In the supervised stage of training, the network is optimized to predict winning human moves from recorded games (Silver et al., 2016). This means that the model initially learns a distribution over actions that reflects strong human play. Later, reinforcement learning through self-play modifies that distribution further by favoring moves that contribute to winning, even when those moves differ from ordinary human preferences.

The relevant subspace in AlphaGo is not static. Training on human games gives the system one way of weighting the move space, but self-play can reshape that weighting by discovering moves that are strategically valuable even if they were initially unlikely under the human-trained policy. This is evident in AlphaGo's famous "Move 37" in game two against Lee Sedol. Google DeepMind later described the move as having roughly a 1 in 10,000 chance of being used under ordinary human expectations, and contemporary commentators initially reacted to it as bizarre or mistaken (AlphaGo website). Yet the move proved strategically powerful. The significance of Move 37 is that AlphaGo algorithm was able to reorganize its relevance subspace. A move that looked negligible from the standard human play became salient within AlphaGo's learned evaluation of the board. This shows that the relevant subspace is not fixed space, but can be reshaped by learning so that actions humans would usually disregard become worthy of search and eventual selection.

The next subsection turns to the value network, which contributes a different kind of relevance signal. If the policy network helps determine which moves are worth considering, the value network helps determine which resulting board states are worth taking seriously as promising futures.

6.3. The Value Network and the Relevance of Future States

The policy network reweights the immediate move space, while the value network reweights possible future positions by estimating how favorable a given board state is for winning the game. In the terms of this paper, the value network helps determine which future trajectories deserve further computational attention. This is important because planning in Go also requires considering how far to continue exploring a line of play. A brute-force search procedure would attempt to simulate many continuations very deeply, but in a game as complex as Go that quickly becomes infeasible. What is needed is some way of deciding when a partially explored line of play already looks promising enough, or unpromising enough. In other words, AlphaGo needs a mechanism that tells it to stop searching the game tree and make a move. The value network addresses a planning problem closely related to what earlier philosophers described as Hamlet’s problem: when to stop deliberating and start acting.

The value network is trained to estimate the probability of winning from a given board state. Rather than searching all the way to the end of the game before assigning significance to a position, AlphaGo can use the value network to approximate the long-term promise of that position earlier in the search process (Silver et al., 2016). This means that the system does not need to treat every future branch as equally worth expanding to terminal depth. Some branches can be taken seriously because they lead to positions with high estimated value, while others can be deprioritized because the resulting board states appear much less favorable.

For example, suppose that from the current board state there are two possible branches of play. The value network estimates that one branch leads to a future position with a relatively high probability of winning, while the other leads to a much less favorable position. This does not mean that the branches have been fully verified and eliminated through exhaustive search. Rather, it means that the system has a learned basis for treating one future as more promising than the other before search reaches the end of the game. In this way, relevance in planning is not only about selecting the next move, but also about determining which possible futures are worth continuing to explore.

The value network plays a relevance-managing role by helping AlphaGo avoid treating depth in the search tree as equally important everywhere. Some lines of play become worth pursuing because they lead to states with strong winning probability, while others recede because the futures they open look strategically weak. We can think of the value network contributing a second form of dynamic reweighting over the space of possible future states.

AlphaGo was not the first game playing system to use evaluations to help with searching the game tree. However, classical game playing systems generally had hand-designed and explicit human heuristics about position or local structure. AlphaGo’s value network instead learns from data to map board states to expected outcomes. That learned evaluation can then be used during search to guide which branches deserve more serious consideration. The result is a closer integration between learning and search than in systems where evaluation and pruning are largely fixed in advance.

From the standpoint of this paper, then, the value network contributes to relevance by helping determine which futures matter enough to justify continued computation. If the policy

network helps identify promising next moves, the value network helps identify promising future states. The next subsection turns to how Monte Carlo Tree Search integrates both signals into a single search process.

6.4. Monte Carlo Tree Search as Guided Search Through a Reweighted Space

The previous two subsections argued that AlphaGo uses the policy network to reweight the space of immediate actions and the value network to reweight the space of future states. The remaining question is how those learned signals are integrated into planning. This is the role of Monte Carlo Tree Search (MCTS). MCTS is the procedure that moves through the game tree, but in AlphaGo it does not operate over a space of equally plausible possibilities. It is guided by the reweighting provided by the policy and value networks.

At a high level, the algorithm of MCTS works by repeatedly traversing the game tree, selecting a path, expanding a new node, evaluating that node, and then propagating the resulting information back through the tree. In AlphaGo, each of these steps is influenced by the learned networks. The policy network helps guide which moves are preferred, while the value network helps estimate the strength of newly reached board states without requiring the search to continue all the way to terminal positions. As a result, the MCTS search procedure does not allocate its effort evenly across the tree. It spends more time in regions that appear strategically promising and less time in regions that appear less useful.

This is important because it distinguishes AlphaGo from earlier game-playing algorithms. Take a game playing algorithm with a brute-force planner. A brute-force algorithm would attempt to expand all legal moves and their continuations as evenly as possible, subject only to computational limits. AlphaGo, instead, combines search with the learned signals from the policy and value networks that indicate which branches of the game tree are worth prioritizing. If a branch begins with a move the policy network rates highly and leads to states the value network evaluates favorably, that branch will receive more search effort. If another branch begins with a low-priority move and leads to weak board evaluations, it will receive less search effort. The result is not the elimination of most of the tree, but a strong concentration of computation in a relatively small region of it.

In this way, MCTS can be understood as guided search through a reweighted space. The search procedure is still crucial. AlphaGo does not pick the highest-probability move from the policy network and stop there. Search is needed to test, refine, and compare possible continuations. But the search is directed by learned estimates that make some parts of the tree much more salient than others.

AlphaGo combines two aspects of relevance management that were separated across the earlier case studies. The language model case relied on contextual reweighting that produced an output distribution over continuations. The probabilistic grammar case relied on an stochastic exploration to move through a large structured hypothesis space. In AlphaGo, relevance is managed by a close interaction between learned evaluation (from the policy and value networks) and explicit exploration (through MCTS).

MCTS does not independently solve the relevance problem in Go. It operationalizes relevance within planning by directing computational effort toward branches of the game tree that have already been made comparatively salient by the policy and value networks. The result is a planning system that can explore a very large space of future possibilities without treating every branch as equally worthy of search.

7. Comparison and Synthesis: Relevance Across Language, Theory Building, and Planning

7.1. A Shared Computational Pressure

This paper examined three case studies from different domains of cognitive science. Large language models operate in the domain of language, where the system must generate context-appropriate continuations and interpretations. Structured probabilistic grammars operate in the domain of theory building, where the system must generate and evaluate candidate explanations for limited data. AlphaGo operates in the domain of planning, where the system must select promising actions and futures within a vast game tree. At the level of their immediate goals, these systems are clearly not doing the same thing.

Despite these differences, all three case studies face the same broad computational pressure. In each case, the system must operate in a problem space that is far too large to treat all possibilities equally. A language model cannot assign equal computational attention to every possible continuation of a sequence. A theory-building system cannot seriously entertain every hypothesis that might explain a dataset. A planning system cannot explore every legal move and every future branch to the end. If successful performance required exhaustive search over all available possibilities, none of these systems would be tractable.

In each case, the difficulty is not only that the system must eventually produce an output. It is that the system must first determine which possibilities deserve further computation and which do not. Language requires the system to privilege some continuations and interpretations over many alternatives. Theory building requires the system to privilege some hypotheses over others. Planning requires the system to privilege some actions and futures over others. Although these are different domains, they share the same underlying challenge: intelligent performance depends on concentrating computation on a relatively small portion of a much larger space.

This shared pressure, what this paper called the relevance problem, is what makes comparison across the case studies theoretically useful. The aim is not to claim that the three systems solve the same task, or that they instantiate one common architecture. Rather, the point is that each of them offers a different computational response to the same general problem: how to make some possibilities matter more than others when the space of possibilities is too large to be handled exhaustively. The next subsection argues that, despite their differences, the three systems converge on a common functional strategy for managing that pressure.

7.2. A Shared Strategy: Dynamic Reweighting

Across all three case studies, the central shared pattern is a common computational strategy. In each case, the system manages the relevance problem by dynamically reweighting its problem space so that a comparatively small, context-appropriate subset of possibilities becomes computationally dominant. This is the main theoretical claim that emerges from the comparison.

The idea of dynamic reweighting is one in which a system does not need to permanently remove most of the possibilities in its environment or hypothesis space in order to act intelligently. Instead, it can change the importance for each of those possibilities for the current task. Some possibilities become highly influential in computation, while others recede into the background. What matters is not that irrelevant possibilities disappear once and for all, but that they do not exert much influence in the present context. This is what allows the system to focus its limited computational resources on what is relevant without losing flexibility when the context changes.

Dynamic reweighting appeared in different forms across the three cases. In large language models, attention reweights contextual information and supports an output distribution in which only a relatively small set of continuations carries significant probability. In structured probabilistic grammars, the space of candidate theories is reweighted by priors and likelihoods so that some hypotheses become much more plausible than others given the observed data. In AlphaGo, policy and value networks reweight the move space and future board states so that some actions and trajectories become far more worthy of search than others. In each case, relevance is managed by reorganizing the computational landscape rather than by treating every possibility as equally important.

The idea of a relevant subspace is also useful here. The relevant subspace is the smaller region of possibilities that becomes computationally salient after reweighting has taken place. In language, this may be a small cluster of likely continuations. In theory building, it may be a small family of hypotheses with substantial posterior support. In planning, it may be a relatively narrow portion of the game tree that receives most of the search effort. What unifies these cases is that successful computation does not operate evenly over the full large space. It operates primarily within a dynamically produced subspace that is relevant to the current task.

An important part of the shared dynamic reweighting strategy is that irrelevant possibilities are assigned low-weight rather than erased. This matters because relevance is context dependent. A continuation, hypothesis, or move that is negligible in one context may become highly important in another. If irrelevant possibilities were permanently deleted, the system would lose the flexibility required to adapt when the task, evidence, or environment changes. Reweighting solves this problem by allowing the system to concentrate computation now while leaving open the possibility that different regions of the space may become salient later.

Taken together, these case studies suggest a common principle. Intelligent performance in large spaces does not require exhaustive evaluation of all possibilities, nor does it require that relevance be handled by one fixed, universal mechanism. What it does require is some way of dynamically reorganizing the space so that a smaller, context-appropriate subset becomes

computationally dominant. That shared strategy is the strongest generalization supported by the comparison developed in this paper.

7.3. Different Mechanisms but Same Functional Pattern

Although the three case studies exhibit a shared strategy of dynamic reweighting, they do not do so through the same mechanism. The argument of this paper is not that large language models, structured probabilistic grammars, and AlphaGo all instantiate one literal computational process nor be reduced to a single unified architecture. The claim is that they share a common functional pattern through importantly different means.

In the large language model case, relevance is managed through attention-based computation. The model reweights contextual information across the input sequence, and the resulting hidden representations support a next-token distribution in which some continuations become much more probable than others. In the theory-building case, relevance is managed through a combination of mechanisms. A grammar structures the hypothesis space, Bayesian inference reweights candidate theories relative to the data, and MCMC explores that reweighted space through local stochastic search. In the planning case, AlphaGo uses its policy and value networks to evaluate moves and future states, while Monte Carlo Tree Search directs computational effort toward the branches made most promising by those learned evaluations.

These differences matter because they show that the shared functional pattern identified in this paper (dynamic reweighting) is not merely architectural. The systems differ in representational format, learning procedure, and mode of computation. Large language models operate over vector representations and sequence prediction. Probabilistic grammars operate over structured symbolic theories. AlphaGo combines learned neural evaluation with tree search in a planning problem. At this level, the systems are not interchangeable. What the comparison reveals is therefore not a single mechanism hidden beneath all three cases, but a common solution to the broader relevance problem. If three very different systems all manage large problem spaces by reorganizing them so that some possibilities become computationally dominant and others recede, then the shared pattern is unlikely to be a coincidence of one particular architecture. Instead, it suggests that dynamic reweighting is a more general strategy for tractable cognition.

Framing the commonality at the functional level also helps avoid a stronger claim that the paper cannot support. The present comparison does not show that the systems share the same learning dynamics, the same internal representations, or the same theory of cognition. Nor does it show that they all capture human relevance judgments in the same way. What it shows is narrower but still important: across language, theory building, and planning, successful systems often manage relevance by dynamically reweighting which possibilities matter most for ongoing computation. That is the level at which the comparison should be understood.

7.4. Different Balances of Constraining and Searching

The comparison across the case studies also helps clarify a second claim of the paper. Although all three systems rely on dynamic reweighting, they differ in how they balance constraining a space and searching through it. This is not a rigid taxonomy as the boundary between constraining and searching is often blurry. A search procedure can itself depend on prior reweighting, and a constraining mechanism may matter only because it guides later exploration. Still, as a comparative dimension, the distinction is useful.

To start, large language models lean most heavily toward constraining the space of possibilities. Through attention and the resulting output distribution, the model makes a relatively small set of token continuations computationally dominant at each step. There is a later decoding search procedure that selects among the weighted possibilities, but compared with the other case studies, the main relevance work occurs in the construction of the next token distribution itself. The system does not explicitly search through candidate continuations in the way that theory-learning or planning systems search through hypotheses or action branches.

Structured probabilistic grammars, in contrast, lean slightly more towards searching the problem space. Here the grammar constrains the overall form of the hypothesis space by determining what kinds of theories can be expressed, but the identification of relevant hypotheses depends much more heavily on search. Bayesian inference reweights the space normatively, and MCMC explores that reweighted space through local stochastic moves. In this case, relevance is managed both by the prior organization of the space and the active exploration within it. The system therefore relies on both constraining and searching, with the search component playing a more prominent role than in the language-model case.

AlphaGo combines both space constraining and searching the most explicitly. Its policy and value networks constrain the planning space by making some actions and future states much more salient than others. Then Monte Carlo Tree Search performs an explicit search through the game tree, revisiting and expanding some branches more than others. Relevance in this case is managed through a tight integration of constraining and searching. Learned evaluation narrows and reshapes the space, and guided search then allocates computational effort within that reshaped space.

Thus, the constraining-searching distinction is best understood as a continuum of emphasis, not as a set of mutually exclusive categories. The language-model case sits closer to the constraining end, the probabilistic-grammar case relies more visibly on search within a structured space, and the AlphaGo case integrates both in a comparatively balanced way. The value of this distinction is not that it yields a perfect classification, but that it helps describe an important way in which systems can differ even while sharing the broader strategy of dynamic reweighting.

7.5. Limits and Open Questions

The comparison developed in this paper is limited in scope. This paper does not show that large language models, structured probabilistic grammars, and AlphaGo share a single underlying mechanism, nor does it establish a unified theory of relevance across all domains of

cognition. The argument is that these systems provide concrete examples of how relevance can be managed in practice when a problem space is too large to be handled exhaustively, and they do so through different forms of dynamic reweighting.

One limitation of this comparison is that probability, value, or weight should not be used as defining relevance in a philosophical sense. In the systems examined here, weighting functions provide a computational indicator of which possibilities are being prioritized for ongoing inference or action. That is informative but it does not tell us about what relevance ultimately is. A low-probability continuation, hypothesis, or move is not automatically incorrect, and a highly weighted possibility is not guaranteed to be completely true. The paper therefore uses relevance in an operational sense: what matters is what the system treats as worth further computation under current conditions.

A second limitation is that the comparison mainly explains how relevance is managed at inference time, not how the capacity for relevance management is learned in the first place. In each case, the system depends on prior structure. Large language models depend on parameters shaped by large-scale training. Probabilistic grammars depend on a predefined representational language and primitives selected in advance. AlphaGo depends on learned policy and value functions shaped by supervised training and self-play. This means that an important question remains unresolved: how does a system come to acquire the weighting functions or representational resources that later allow it to make the right information relevant? Showing how a trained or prestructured system manages relevance does not by itself explain how that capacity originally develops.

This issue is important for the broader goals of cognitive science. The comparison offered in this paper should be understood as identifying computational strategies to manage relevance rather than a complete account of how people learn what matters in the world. Human relevance sensitivity develops over time, across tasks and environments, and is shaped by perception, action, memory, and social interaction. The systems discussed here may illuminate some general computational principles, but they do not yet amount to a full theory of human relevance judgments.

Closely related, a third limitation concerns the role of prior structure. The systems examined here do not manage relevance from a completely unconstrained starting point. Each depends on architectural, representational, or training biases that already shape which possibilities are expressible and how they are weighted. This is especially clear in the probabilistic grammar case, where the primitives of the representational language are fixed in advance, but it is also true of the neural-network cases, where architecture and training data strongly influence what patterns can later be treated as salient. For that reason, these case studies should not be read as showing that the relevance problem has been solved once and for all. Rather, they show how successful systems can manage relevance given particular forms of prior organization.

Even with these limitations, the comparison remains theoretically useful. It shows that the relevance problem should not be understood only as a philosophical challenge or as evidence that

tractable cognition is impossible. Across several distinct domains, successful systems already demonstrate concrete ways of concentrating computation on a manageable portion of a much larger space. The present paper does not claim to close the question of relevance. It aims instead to clarify one important lesson from existing computational work. Intelligent systems often succeed not by considering everything, but by dynamically reorganizing what matters for the task at hand.

8. Conclusion

This paper examined whether current computational frameworks of cognitive science offer useful insights into the relevance problem. I argued that they do, not because they provide a single unified solution, but because they show concrete ways in which successful systems manage large problem spaces without exhaustive search. Across language modeling, theory building, and planning, the systems examined here all face the same broad computational pressure: they must identify which possibilities deserve further computation while ignoring many others that are available in principle.

The main claim of the paper is that these systems converge on a shared functional strategy. In each case, the system dynamically reweights its problem space so that a smaller, context-appropriate subset of possibilities becomes computationally dominant. In large language models, this occurs through attention-based reweighting of contextual information and the resulting output distribution over continuations. In structured probabilistic grammars, it occurs through the combination of a structured hypothesis space, Bayesian reweighting, and stochastic search. In AlphaGo, it occurs through learned policy and value networks that guide search through a large planning space. What the three systems share is therefore not one mechanism, but a common way of managing relevance under constraint.

At the same time, the comparison also reveals an important limitation. Showing how a system manages relevance at inference time does not yet explain how it acquires the representational resources and weighting functions that make such management possible. That question remains open, especially for any broader theory of human cognition. For that reason, this paper does not claim that the relevance problem has been solved once and for all. Rather, it argues that successful computational systems already provide mechanistically explicit examples of how relevance can be managed in practice.

The relevance problem should not be treated only as a philosophical obstacle. It can also function as a productive comparative lens for understanding what successful cognitive systems must do. They must find ways to concentrate limited computation on what matters now, while retaining the flexibility to shift when the context changes. The case studies examined here suggest that dynamic reweighting is one important strategy by which that is achieved.

References

AlphaGo. Google DeepMind. <https://deepmind.google/research/alphago/>

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in neural information processing systems*.

Campbell, M., Hoane Jr, A. J., & Hsu, F. H. (2002). Deep blue. *Artificial intelligence*.

Castillo, L., León-Villagr , P., Chater, N., & Sanborn, A. N. (2024). Explaining the flaws in human random generation as local sampling with momentum. *PLOS Computational Biology*.

Clark, A. (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*.

Darling, T., Corcoran, A. W., & Hohwy, J. (2025). Solving the relevance problem with predictive processing. *Philosophical Psychology*.

Dasgupta, I., Schulz, E., Goodman, N. D., & Gershman, S. J. (2018). Remembrance of inferences past: Amortization in human hypothesis generation. *Cognition*.

Dasgupta, I., Schulz, E., Tenenbaum, J. B., & Gershman, S. J. (2020). A theory of learning to infer. *Psychological Review*.

Dennett, D. C. (1990). Cognitive wheels: The frame problem of AI. *The philosophy of artificial intelligence*.

Fodor, J. (1987). Modules, frames, fridgions, sleeping dogs, and the music of the spheres.

Fodor, J. (1983). The Modularity of Mind. *The MIT Press*.

Friston, K. (2010). The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*.

Gershman, S. J., & Goodman, N. D. (2014). Amortized inference in probabilistic reasoning. *Proceedings of the 36th Annual Conference of the Cognitive Science Society*.

Goodman, N. D., & Frank, M. C. (2016). Pragmatic language interpretation as probabilistic inference. *Trends in Cognitive Sciences*.

- Griffiths, T. L., Lieder, F., & Goodman, N. D. (2015). Rational use of cognitive resources: Levels of analysis between the computational and the algorithmic. *Topics in Cognitive Science*.
- Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*.
- Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*.
- Lieder, F., & Griffiths, T. L. (2020). Resource-rational analysis: Understanding human cognition as the optimal use of limited computational resources. *Behavioral and Brain Sciences*.
- McCarthy, J. & Hayes, P.J. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence 4*.
- McDermott, D. (1986). We've Been Framed: Or, Why AI Is Innocent of the Frame Problem.
- Nakatani, H., & Yamaguchi, Y. (2014). Quick concurrent responses to global and local cognitive information underlie intuitive understanding in board-game experts. *Scientific Reports*.
- Sanborn, A. N., & Chater, N. (2016). Bayesian brains without probabilities. *Trends in Cognitive Sciences*.
- Sanborn, A. N., & Chater, N. (2017). The sampling brain. *Trends in Cognitive Sciences*.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*.
- Sperber, D., & Wilson, D. (2004). Relevance theory. *Handbook of Pragmatics*. Oxford: Blackwell.
- Tenenbaum, J. B., Kemp, C., Griffiths, T. L., & Goodman, N. D. (2011). How to grow a mind: Statistics, structure, and abstraction. *Science*.
- Ullman, T. D., Goodman, N. D., & Tenenbaum, J. B. (2012). Theory learning as stochastic search in the language of thought. *Cognitive Development*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*.

Zhu, J. Q., Sundh, J., Spicer, J., Chater, N., & Sanborn, A. N. (2024). The autocorrelated Bayesian sampler: A rational process for probability judgments, estimates, confidence intervals, choices, confidence judgments, and response times. *Psychological Review*.