Game of Thought: Robust Information Seeking with Large Language Models Using Game Theory

Anonymous submission

Abstract

Large Language Models (LLMs) are increasingly deployed in real-world scenarios where they may lack sufficient information to complete a given task. In such settings, the ability to actively seek out missing information becomes a critical capability. However, existing approaches to enhancing this ability often rely on simplifying assumptions that degrade *worst-case* performance. This is an issue with serious implications in high-stakes applications. In this work, we introduce and formalize the Strategic Language Search (SLS) problem along with its variants as a two-player zero-sum extensive form game. This formulation provides a principled framework for evaluating the information-seeking capability of LLMs. We propose Game of Thought (GoT), a simple yet effective framework that applies game-theoretic techniques to approximate a Nash equilibrium strategy for the restricted variant of the game. Empirical results demonstrate that our approach consistently improves worst-case performance compared to (1) direct prompting-based methods and (2) heuristic-guided search methods across all tested settings.

1 Introduction

Large language models (LLMs) are increasingly being deployed in high-stakes environments like planning (Zhang et al. 2024), medical diagnosis (Li et al. 2024), as well as other tasks exhibiting partial observability (Li, Kim, and Wang 2025). In such environments, the LLM may not have sufficient information to complete the its assigned task. This necessitates an information-seeking process, typically via the use of clarification questions. To quantitatively assess an LLM's information-seeking ability, we turn to the *Game of 20 Questions* (Siegler 1977). Here, an item is first chosen from a known set. The player then sequentially asks up to twenty Yes/No questions about the item, aiming to identify the item using as few questions as possible.

Information seeking often requires some degree of lookahead. Well-known methods involve a combination of prompting and LLM-based reasoning, such as Self-Consistency (SC) (Wang et al. 2022) and Tree of Thought (ToT) (Yao et al. 2023). More recently, Hu et al. (2024) propose Uncertainty of Thought (UoT), which utilizes tree search to *explicitly model and optimize* for the information gained from clarification questions. However, their approach assumes that the item is chosen uniformly at random. This assumption is unlikely to hold in the real-world, either because (i) the application itself does not naturally admit a probabilistic interpretation, or (ii) even if such a distribution existed, it might not be easily obtained, and likely not uniform. Particularly for high-stakes environments, we argue that one should assume that the *worst-case* item is chosen. That is, the item is chosen *adversarially*, and our goal is to utilize a questioning strategy that maximizes the worst case performance, regardless of how the item is chosen.

Motivated by this, we propose Game of Thought (GoT), a game-theoretic framework designed to handle such adversarial environments. We model the information-seeking process as a two-player, zero-sum extensive form game with imperfect information. In this formulation, the item (distribution) is assumed selected by an adversary seeking to impede the information seeker. GoT then optimizes for the best worst-case performance by approximating the Nash equilibrium of this game, avoiding any need for strong prior assumptions on item distribution.

Contributions **(i)** We formulate the Strategic Language Search (SLS) problem and its variants. **(ii)** We establish game theoretic solutions that are able to solve the SLS and its variants optimally, and highlight that these strategies are necessarily randomized. **(iii)** We propose GoT to approximate the Nash equilibrium in the SLS-restricted variant. **(iv)** Empirically, we demonstrate that GoT is superior to (a) direct prompting and (b) UOT-based methods in the worst case. The improvements over these methods become more significant in the more realistic weighted variant of SLS.

2 Related Work

Uncertainty of Thought GoT can be regarded as an extension of *Uncertainty of Thought* (UoT) by Hu et al. (2024). The authors propose performing depth limited search over possible questions asked, seeking to maximize expected information game under the assumption that the items are *chosen uniformly at random*. GoT performs similar explicit lookahead, but obviates this assumption applies game solvers to optimize for the best worst-case item chosen.

Two Player Zero-Sum Games Game of Thought models the robust information-seeking problem as a two-player zero-sum *imperfect information* extensive form games (Shoham and Leyton-Brown 2008). Scalable game solvers

have led to strong or even superhuman performance in various recreational games such as poker (Brown and Sandholm 2018, 2019; Moravčík et al. 2017), starcraft (Vinyals et al. 2019), stratego (Perolat et al. 2022), diplomacy (FAIR), dark chess (Zhang and Sandholm 2021), as well numerous real-world security applications (Jain, An, and Tambe 2013; Pita et al. 2008; Shieh et al. 2012; An, Tambe, and Sinha 2017). Recently, Liu et al. (2025) show that fine-tuning LLMs with simple zero-sum games leads to improved performance in several reasoning benchmarks.

Information-seeking ability of LLMs Prior studies examined the information-seeking ability in the context of ambiguous queries or under-specified tasks, and utilize clarification questions to identify either user intent (Kuhn, Gal, and Farquhar 2022; Zhang et al. 2024; Xu et al. 2019) or elicit user preferences (Handa et al. 2024; Li et al. 2023). In such contexts, it can be roughly seen as a reasoning task, where many well-established methods have shown that exploration (Wang et al. 2022; Yao et al. 2023) can improve LLMs' abilities. Other methods are primarily motivated by heuristics such as information gain (Grand et al. 2024; Piriyakulkij, Kuleshov, and Ellis 2023), similar to UoT (Hu et al. 2024). Various benchmarks for evaluating information-seeking ability of LLMs have been constructed in the context of underspecified tasks (Li, Kim, and Wang 2025) and medical diagnosis (Li et al. 2024). We do not explicitly evaluate GoT in these contexts, but believe that it can be applied to them with some modifications.

3 Problem Formulation

A Strategic Language Search (SLS) game is played over a finite set $\mathcal{S}$ of n distinct items. The game is played between two competitive (zero-sum) players, the *Answerer* and *Questioner*. We will assume a countable set of binary (i.e., true-false) questions $\mathcal{Q}$. Each question is uniquely specified by a finite-length natural language string. For every item $s \in \mathcal{S}$, we denote by $f : \mathcal{Q} \times \mathcal{S} \to \{0, 1\}$ the *answer* to $q \in \mathcal{Q}$ for some $s \in \mathcal{S}$. A natural but degenerate $\mathcal{Q}$ is one which we denote by $\mathcal{Q}_\infty$. A possible textual representation of $\mathcal{Q}_\infty$ would be the set of questions stated in boolean form "Is the Item 1, Item 3, or Item 5, but not Item 2 and 4?"

A SLS is defined by $(\mathcal{S}, \mathcal{Q}, f)$ and proceeds in two phases. In the first phase, the Answerer privately selects a single item $s^* \in \mathcal{S}$. In the second phase, the Questioner elicits information about s^* by sequentially asking and obtaining answers to a series of questions $q_1, a_1, \ldots, q_T, a_T$ and where $q_t \in \mathcal{Q}$ and $a_t = f(q_t, s^*)$, i.e., the answer to q_t for s^*.

The history of questions and answers up to and including time $t \geq 0$ is given by $H_t = (Q_t, A_t)$, where $Q_t = (q_1, \ldots, q_t), A_t = (a_1, \ldots, a_t)$ such that $Q_t(\tau) = q_\tau$ and $A_t(\tau) = a_\tau$. Given history H, we denote by $H'(q, a)$ the new history when a new question q is asked with answer a. The Questioner selects its questions online, i.e., q_t may depend on its observed history H_{t-1}. The length of history H is denoted by $|H|$. The set of items consistent with some observed history $H = (Q, A)$ is denoted by

$$S(H) = \{s \in \mathcal{S} | \forall \tau \in [|H|], f(Q(\tau), s) = A(\tau)\} \subseteq \mathcal{S}.$$

Note that $S(H)$ is never empty, since it must contain at least s^*. The game continues until time T, when the Questioner can identify s^* with certainty, i.e., $|S(H)| = 1$, after which the game ends with the Answerer (resp. Questioner) incurring a total reward (resp. cost) of $|H|$. The objective is to find a strategy (or equivalently, policy) for the Questioner to minimize its expected cost incurred regardless of how the Answerer chooses s^*. This strategy corresponds to the *Nash equilibrium* of a zero-sum game and will typically be randomized. We will define the solution concept formally after stating several assumptions and variants.

Assumption 1. *Each question-item pair $(q, s) \in \mathcal{Q} \times \mathcal{S}$ has a definite, unique answer $f(q, s)$ independent of history.*

Assumption 2. *After fixing s^*, the Answerer cannot lie with regard to its answers to questions.*

Assumption 3. *$\mathcal{S}$, $\mathcal{Q}$ and f are common-knowledge.*

Assumption 4. *We have constant time oracle access to f.*

Assumption 5. *For every pair of distinct items $s, s' \in \mathcal{S}$, there exists some $q \in \mathcal{Q}$ where $f(q, s) \neq f(q, s')$.*

Assumptions 1 and 2 are implicit, informal assumptions baked into the definition of f while Assumption 5 is a technical assumption imposed on $\mathcal{Q}$ and f. It is needed to ensure that there exists a strategy that allows the Questioner to find the correct item in finite time. Note that Assumption 5 is trivially satisfied when $\mathcal{Q}$ includes "identity" questions such as "is the item equal to s?".

Example 1. *Consider the SLS with $\mathcal{S}$ and $\mathcal{Q}$:*

$$\left\{ \begin{array}{l} s^{(1)} : \text{'Oppenheimer'} \\ s^{(2)} : \text{'Alan Turing'} \\ s^{(3)} : \text{'A Beautiful Mind'} \end{array} \right\}, \left\{ \begin{array}{l} q^{(1)} : \text{'Related to codes?'} \\ q^{(2)} : \text{'Is it a movie?'} \\ q^{(3)} : \text{'Is it a person?'} \end{array} \right\}$$

Then $f(q^{(i)}, s^{(j)}) = 0$ if $j = i$, 1 otherwise.

It is easy to verify that Assumption 5 holds in Example 1. Clearly, by selecting $q^{(1)}$ and $q^{(2)}$ in sequence, one can guarantee that the Questioner asks no more than 2 questions. The best the Answerer can do against such a strategy is to select $s^* = s^{(2)}$ or $s^{(3)}$, since doing so guarantees that there are always two items consistent after the first question, i.e., $|S(H_1)| = 2$. However, the Questioner can do better by choosing the first question uniformly at random. This guarantees that regardless of the Answerer's choice of s^*, there is $1/3$ probability that the item can be determined with exactly one question, thus the expected number of questions asked is $1/3 \cdot 1 + 2/3 \cdot 2 < 2$. This strategy minimizes the worst-case cost regardless of what s^* the Answerer chose. In this example, we are lucky that the optimal strategy for the Questioner is symmetric due to the circular symmetric nature of $\mathcal{Q}$ and f. This is not true for more general $\mathcal{Q}, f$.

So far, SLS has been presented mostly as a combinatorial problem. Indeed, when $\mathcal{Q}$ is finite, we have:

Theorem 1. *Given a known (deterministic) s^*, deciding if there is a sequence of k questions $Q \subseteq \mathcal{Q}$ such that $S(H) = \{s^*\}$ is NP-complete.*

The reduction is straightforward from set-cover and included in the appendix. Indeed, a very similar problem known as *teaching dimension* was studied by Goldman and Kearns (1995), and is in turn closely related to other related concepts in learning theory such as the VC-dimension. In game theoretic parlance, this implies that the *best-response* of the Questioner to any deterministic Answerer strategy is hard to compute.

Theorem 2. *Let $(\mathcal{S}, \mathcal{Q}_\infty, f)$ be a SLS, where $|\mathcal{S}| = 2^k$ for some positive integer k. An "even-split" strategy where $\forall t, q_{t+1}$ is selected such that $|S(H'_t(q_{t+1}, 0))| = |S(H'_t(q_{t+1}, 1))|$ minimizes the number of questions asked in the worst case for the Questioner and costs exactly k questions.*

Theorem 2 states that in the special case where $\mathcal{Q} = \mathcal{Q}_\infty$ the problem becomes easy, agreeing with the intuition that "binary search" could be optimal. We show in the appendix that the even-split strategy is optimal in the UoT where s^* is chosen uniformly at random (Hu et al. 2024); in fact, they constitute a Nash equilibrium in game setting. This justifies their implicit approach of maximizing entropy loss and usage of LLM prompts that encourage even splits.

SLS-Restricted (SLSR) Vanilla SLS can be quite cumbersome to reason about for larger n and $\mathcal{Q}$. As such, we propose a SLSR, a restricted variant where questions are generated (typically from a language model) based on the remaining items $S(H)$. A SLSR is formally given by $(\mathcal{S}, \mathcal{Q}, f, g)$, where $g : 2^{\mathcal{S}} \setminus \phi \to 2^{\mathcal{Q}}$ is a set function taking a nonempty set of items $S \subseteq \mathcal{S}$ and outputs a nonempty set of no more than $m \geq 1$ questions; here m is a parameter associated with g. The rules and payoffs in SLSR are identical to SLS, except that the Questioner is restricted to selecting $q_t \in g(S(H_{t-1}))$. To ensure that progress can always be made, we impose a stronger version of Assumption 5 and require that g outputs at least one question that strictly reduces the set of consistent items.

Assumption 6. *For every subset $S \subseteq \mathcal{S}$, there exists some $q \in g(S)$ for and a pair of distinct items $s, s' \in S$ where $f(q, s) \neq f(q, s')$*

Weighted-SLS (WSLS) In many real-world scenarios, certain items carry greater importance than others and should be prioritized during the information-seeking process. For instance, in medical diagnosis, delays in identifying life-threatening conditions can have more severe consequences. Hence, we propose weighted-SLS, a variant of SLS defined by $(\mathcal{S}, \mathcal{Q}, f, w)$, where $w : \mathcal{S} \to \mathbb{R}^+$. The cost incurred by the Questioner with s^* as the selected item is redefined to be $w(s^*) \cdot |H|$, imposing greater penalties for prolonged interactions involving high-weight items. Weighted SLSR (WSLSR) is defined in a similar manner by $(\mathcal{S}, \mathcal{Q}, f, g, w)$.

Example 2. *Define the SLSR $(\mathcal{S}, \mathcal{Q}, f, g)$ where $\mathcal{S}, \mathcal{Q}, f$ are identical to Example 1 and $g(S) = \{q^{(1)}, q^{(2)}\}$ when $S = \mathcal{S}$ and $\mathcal{Q}$ otherwise.*

In Example 2, the first question is restricted to be either $q^{(1)}$ or $q^{(2)}$. Regardless of how the Questioner randomizes between them, the Answerer can always select $s^* = s^{(3)}$; this guarantees that $a_1 = 1$ and hence $|S(H_1)| = 2$. Thus, the Questioner requires at least one additional question, taking 2 questions in total.

Example 3. *Define the WSLS $(\mathcal{S}, \mathcal{Q}, f, w)$ with $\mathcal{S}, \mathcal{Q}, f$ identical to Example 1 and $w(s^{(1)}) = 3, w(s^{(2)}) = w(s^{(3)}) = 2$.*

In Example 3, the Questioner can chose q_1 to be equal to $q^{(1)}, q^{(2)}, q^{(3)}$ with probability $3/4, 1/8, 1/8$. This ensures the expected cost is no greater than $15/4$. In contrast, choosing q_1 uniformly incurs a cost of 5 when the Answerer chooses $s^* = s^{(1)}$.

Remark. *In SLS and its variants, the game only ends when the Questioner is perfectly sure of the s^*. Another alternative formulation permits the Questioner to explicitly guess the item at any stage, with penalties for incorrect guesses. We chose the current formulation since it captures the constraints in high-stakes environments better, e.g., in medical diagnosis, one wishes to rule out all other possibilities before moving on to treatment. Roughly speaking, our formulation penalizes incorrect guesses with "infinite" penalty.*

Defining SLS via Large Language Models. For many applications, $\mathcal{S}$ is too large for us to realistically hand-curate a set of $\mathcal{Q}$, noting that we would like to avoid "unnatural" questions that involve long, complicated boolean expressions. Therefore, in this paper we define $\mathcal{Q}$, f, and g implicitly using Large Language Models (LLM). Specifically, $\mathcal{Q}$ is the set of questions that a LLM can propose asking based on certain prompts, and f is the response of a LLM to a prompt of q when asked about s^*. For instance, `Is Oppenheimer a movie?`. Furthermore, g in SLSR and WSLSR can be defined as the output of a LLM with respect to a prompt asking for $m \geq 1$ questions after defining the rules of SLS and $S(H)$, the items remaining. An example of this is `We are playing a game with the following rules <Rules of SLSR/WSLSR>. Currently the set of possible items is {Oppenheimer, Alan Turing}. Propose m best questions the Questioner can ask.`

Assumption 7. *The LLM for f makes no mistakes when used as an oracle for $f(q, s^*)$.*

Assumption 7 and 1 enable the practical usage of LLMs as a black-box for defining SLS. For most LLMs, Assumption 4 is also satisfied, though the constant factor may be very large. We discuss all assumptions made in the appendix.

4 SLS as Zero-sum Extensive Form Games

A vanilla SLS $(\mathcal{S}, \mathcal{Q}, f)$ may be expressed as a two-player zero-sum extensive form game (EFG) with imperfect information. EFGs are rooted game trees, where each vertex corresponds to the game's entire history. Edges represent actions available to the player-to-move at that vertex. EFGs are endowed with information sets (infosets), which partition vertices belonging to the same player; vertices in the same infoset are indistinguishable to the player owning them, this captures the notion of imperfect information.

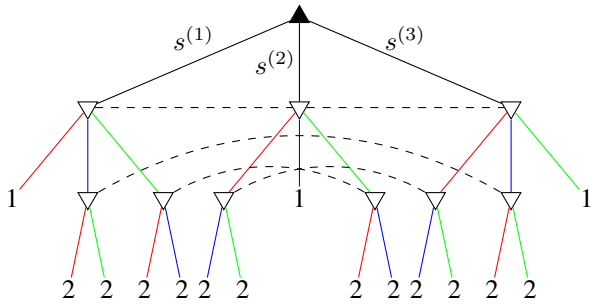

Figure 1: EFG representation of Example 1. The root node ▲ belongs to the Answerer and the other nodes ▽ belong to the Questioner. Edges in black correspond to the choice of item s^*. Colored edges in red, blue, and green respectively refer to questions $q^{(1)}$, $q^{(2)}$, and $q^{(3)}$ respectively. Vertices connected by dotted lines belong to the same infoset. Payoffs (resp. costs) to the Answerer (resp. Questioner) are shown in the leaves. We omit edges (and their descendant subtrees) where the same question is asked more than once; these are never optimal and correspond to dominated actions.

In the EFG formulation, the Answerer only takes an action at the root, after which the Questioner asks up to $n-1$ questions in sequence.[1] Every state apart from the root can be uniquely identified by the tuple (s^*, H) which describes item s^* chosen and a (potentially empty) history of past questions and answers. Leaf vertices are those where either (i) $|H| = n-1$, where the maximum number of questions are asked, or (ii) $|S(H)| = 1$, where s^* is identified. The leaf the rewards (resp. cost) to the Answerer (resp. Questioner) is $|H|$. All non-leaf vertices with the same history H belong to the same information set, which we denote by $I(H)$. Figure 1 illustrates the SLS for Example 1.

A deterministic strategy π of the questioner is given by a mapping from every infoset $I(H)$ to some question $q \in \mathcal{Q}$ to be chosen for $q_{|H|+1}$. Following π when s^* is chosen yields the following history $H_0, \dots H_T$, given by

$$q_{t+1} = \pi(I(H_t)), a_{t+1} = f(\pi(I(H_t)), s^*),$$

where T is the time the game ends, i.e., $|S(H_T)| = 1$. The corresponding utility for the Answerer (resp. cost for the Questioner) at the end of the game is $u(\pi, s^*) = |H_T|$.

Let the set of all deterministic strategies be Π, Δ_Π the probability simplex over Π, $\Delta_\mathcal{S}$ the probability simplex over $\mathcal{S}$. Then the expected utility under random policies $x \in \Delta_\Pi$ and $y \in \Delta_\mathcal{S}$ for Questioner and Answerer respectively is (with some abuse of notation) denoted by

$$u(x, y) = \mathbb{E}_{\pi \sim x, s^* \sim y} \left[u(\pi, s^*) \right].$$

The Nash equilibrium (NE) of this zero-sum game is given by the solution to the bilinear saddle-point problem

$$\min_{x \in \Delta_\Pi} \max_{y \in \Delta_\mathcal{S}} u(x, y) = \max_{y \in \Delta_\mathcal{S}} \min_{x \in \Delta_\Pi} u(x, y) \tag{1}$$

where equality holds as a consequence of Von Neumann's minimax theorem (v. Neumann 1928). The solution (x^*, y^*) yields optimal randomized strategies for each player, i.e., each player is best-responding to the other. In particular, x^* is the *distribution* over Questioner policies that is robust against the worst case choice of s^*. EFGs for SLSR, WSLS and WSLSR may also be constructed similarly by one or both of (i) restricting the actions available at each infoset to $g(S(H))$ or (ii) adjusting leaf payoffs to depend on w and s^*.

Solving EFGs. In general, two-player zero-sum EFGs with perfect recall (including SLS and its variants) can be solved in time polynomial to the size of the game tree. Some classical methods include linear programming (Von Stengel 1996) and counterfactual regret minimization (Zinkevich et al. 2007). Note that the definition of strategy given for the Questioner was in normal (also known as strategic or matrix) form. This leads to an exponentially sized action space, so in practice solvers usually operate (implicitly) in the strategically equivalent behavioral or *sequence form* (Von Stengel 1996). Efficient game-solving and strategy representation in EFGs is a well studied topic and beyond the scope of this paper, though we give a brief overview in the appendix. Instead, we will simply employ off-the-shelf solvers (Liu, Farina, and Ozdaglar 2024; Lanctot et al. 2019). As it turns out, the bottleneck in our proposed solution is not game solving but LLM queries for f and g.

5 Game-of-Thought and Subgame Search

For a SLS game tree with maximum depth D (with a lower-bound[2] of $\log_2(|\mathcal{S}|)$), the total number of information sets is $O((2|\mathcal{Q}|)^D)$. This leads to the number of infosets growing at a rate of $O((2|\mathcal{S}|-2)^{\log_2|\mathcal{S}|})$,[3] which is far too large to reason about even for a relatively small $|\mathcal{S}|$. For practical reasons, we study the Restricted variants of SLS and WSLS with $|g(S(H))| = m = 3$, limiting the number of infosets to $O(|\mathcal{S}|^{2.59})$. Nonetheless, constructing the full (W)SLSR game tree explicitly and solving it is still unpractical, since this requires querying a LLM at each infoset to generate questions and answers, with each query typically incurring a latency of several seconds.

To address this challenge, GoT employs an iterative, on-demand strategy construction paradigm, computing strategies only when an information set is visited. Inspired by the subgame search techniques used in chess and poker AI, GoT constructs a subgame rooted at the current information set and truncates it to a fixed depth d. The leaf nodes of this subgame are evaluated using heuristic functions. This truncated subgame is then solved to obtain a local strategy for the questioner, which informs the selection of the next question and determines the transition to the subsequent information set. An overview of this approach is provided in Figure 2.

[1] This keep the game tree finite, and is justified since asking the same question more than once is suboptimal.

[2] This is a rather optimistic lowerbound, which occurs only when every every $q \in \mathcal{Q}$ splits the items evenly. In most cases $D > \log_2(|\mathcal{S}|)$.

[3] $|\mathcal{Q}| \geq |\mathcal{S}| - 1$ due to Assumption 5

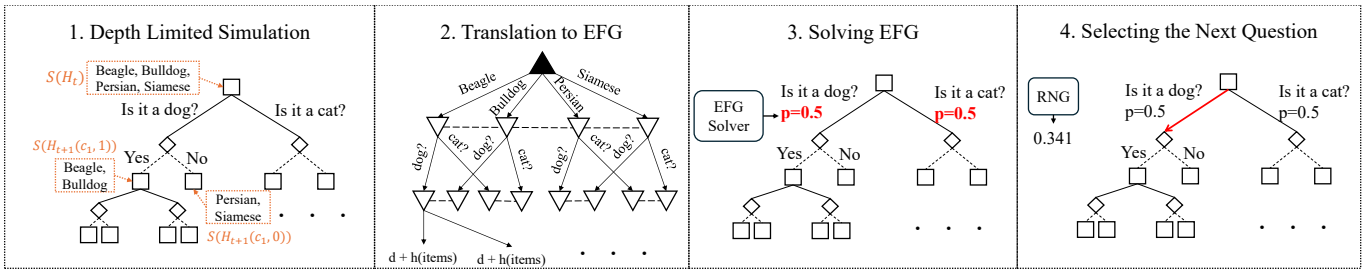

Figure 2: **An overview of GoT.** To choose the question at time step t, we (1) Explore the possible outcomes and future questions (2) Construct the truncated EFG tree (3) Solve the game for a local strategy (4) Use that strategy to choose the question. Steps 1 to 4 are repeated until we reach the end of the game. Steps 1 to 3 are used to devise the strategy, while step 4 is playing the actual (W)SLS(R) game.

1. Depth Limited Simulation We begin by exploring the possible future outcomes and questions for the Questioner necessary to construct the truncated subgame. At infoset $I(H_t)$, assuming the game has not ended, the set $g(S(H_t))$ is generated using a LLM. For each $c_i \in g(S(H_t))$, a LLM is used to identify the set of items for which the answer to c_i is yes denoted by $Y(S(H_t), c_i)$, where

$$Y(S, c_i) = \{s \in S | f(c_i, s) = 1\} \text{ for some } S \subseteq \mathcal{S}.$$

and the complement set denoted by $\bar{Y}(S(H_t), c_i)$, defined in a similar manner. This pair of sets represents the two possible outcomes of asking the candidate question c_i at step t. The simulated histories $H'_t(c_i, 1)$, $H'_t(c_i, 0)$ and the corresponding infosets $I(H'_t(c_i, 1))$, $I(H'_t(c_i, 0))$ are constructed based on this pair of sets. This process is repeated to recursively construct the infosets for up to $t + d$ steps, resulting in a simulation tree as seen in step 1 of Figure 2.

2. Translation to EFG Next, we construct the two-player zero-sum extensive form subgame based on the simulation tree. The tree is converted into an EFG representation of the truncated subgame, as shown in step 2 of Figure 2. In particular, the tree is augmented with the Answerer choosing one of the remaining items from $S(H_t)$, even if $I(H_t)$ is not at the beginning of the SLSR game. Each leaf node in the game tree l is assigned a payoff of $d(l) - 1 + h(l)$ for the Answerer (and the negative of that for the Questioner) where $d(l)$ is the depth of node l in the tree representing the number of questions asked after step t before reaching the node l, and h is a heuristic function used to estimate the remaining number of questions that remain to be asked for the Questioner to identify the correct item. In our experiments we set $h(l) := \log_2(|S(l)|)$ where $S(l)$ is the set of items remaining at leaf node l.

3. Extensive Form Game Solving To solve the resulting extensive form subgame, we used LiteEFG's (Liu, Farina, and Ozdaglar 2024) implementation of counterfactual regret minimization (Zinkevich et al. 2007) to obtain an approximate of the Nash Equilibrium for the Questioner.

4. Selecting the Next Question To choose the question to ask at step $t + 1$, one question q_{t+1} is sampled from $g(S(H_t))$ using the resulting randomized strategy for the Questioner. We move to the next infoset $I(H'_t(q_{t+1}, 0)$ if $f(q_{t+1}, s^*) = 0$, and $I(H'_t(q_{t+1}, 1)$ otherwise.

This process is repeated until an infoset $I(H_n)$ where $|S(H_n)| = \{s^*\}$ is reached.

At first glance, allowing for the Answerer to "re-choose" a new distribution over s^* every iteration in Step 2 seems to accord the Answerer significantly more power, since it would be able to "switch items" depending on the questions asked in actual play. In fact, what we have done is a simple implementation of *maxmargin resolving*, a crucial step in performing *safe* subgame search (Moravcik et al. 2016) in general zero-sum EFGs. Safety implies a performance guarantee with respect to some blueprint strategy or state value estimate. Note that due to the simplicity of SLS, newer techniques and extensions (Brown and Sandholm 2017; Zhang and Sandholm 2021) reduce to maxmargin resolving here.

Theorem 3. *GoT is safe with respect to value estimates that only depend on $S(H_t)$.*

Since safety follows directly from maxmargin search, we defer details to the appendix. In practice, we see a huge improvement over unsafe variants of subgame search.

6 Experiments and Results

We are mainly interested in answering the following:

*Does GoT improve the **worst case** performance compared to other methods in SLSR and WSLSR games?*

Secondarily, we examine how the (i) quality of the questions (ii) simulation depth d, and (iii) difficulty of the dataset affect the performance of GoT.

Experiment Setup

Due to space constraints, specific details and prompts are deferred to the appendix.

Datasets For SLSR and WSLSR, each dataset is a collection of distinct items. Prior studies (Bertolazzi et al. 2023; Zhang, Lu, and Jaitly 2024) used datasets such as *Things* (Hebart et al. 2019) and *Celebrities*, which we found to be too large for this study. This led to us constructing several datasets: **(i) COMMON+**: A set of 136 items built upon the COMMON dataset collected for UoT(Hu et al. 2024), which we further modified by adding more items. **(ii) Breeds**: A

set of 25 popular cat and dog breeds collected by us. **(iii) Skewed 128**: A set of 128 items, consisting of 2 weapons, 6 scientists, 24 dishes, and 96 animals collected by us.

Baselines We primarily compare with UoT (Hu et al. 2024). We additionally consider Direct Prompting (DP) where the LLM is allowed to directly generate the next question, and Direct Choice (DC) where the LLM is asked to choose from a set of candidate questions. Each strategy is played against all possible s^* in each dataset and the worst performance across all items is taken.

Models and Prompts We mainly experimented on GPT 4.1(OpenAI 2024) (`gpt-4.1-2025-04-14` checkpoint) and Qwen 2.5 72B Instruct (Yang et al. 2024). We utilized two different prompts for sampling questions for SLSR. The *even* prompt, adopted from (Hu et al. 2024), explicitly instructs the LLM to ask questions that splits items as evenly as possible. When using even prompts, we noticed unnatural questions (e.g., Does this item begin with a letter from A-M?). This is ill-suited for actual applications as the respondent may find it difficult to answer such questions. This leads to the *natural* prompt, which instructs the LLM to refrain from asking such questions. This led to fewer unnatural questions.

Accounting for Randomness **(1)** When a LLM is used as g to generate questions, the set of candidate questions $\mathcal{Q}(S(H_t))$ may differ across runs. To ensure fair comparisons across methods, we first play the game using GoT and cache the questions used in the simulation tree. The cached questions are reused for UoT and DC. **(2)** The Questioner strategy designed by GoT is nondeterministic. To obtain the performance for each s^*, we take the average over five plays of the game using the same strategy.

SLSR

As seen in Table 1, GoT provides a consistent, albeit minor, improvement over UoT in all settings, improving the worst-case performance by around one turn. Prompting-based DP occasionally outperform UoT, indicating that although UoT can enhance average performance, it may do so at the expense of degraded worst-case performance.

As mentioned, when using the *even* prompt, we occasionally observe "unnatural" questions being sampled that are often effective at splitting item sets into even halves. We suspect this to be why a noticeable improvement in DP can be observed when *even* is used over *natural*. This is less noticeable in GoT and UoT, possibly due to both performing planning via lookahead. This could reduce the influence of the few "unnatural" questions on the overall performance.

From Theorem 2, we know that if a LLM is able to propose questions that perfectly splits $S(H)$ into even halves, there is no need to perform any planning since choosing any of the questions is optimal. However, we see that this is not the case: all methods fall short of the theoretical optimal performance of $\log_2(|\mathcal{S}|)$ by around 2 to 3 turns of interaction. Out of the three datasets, Skewed 128 is intentionally designed to guide LLMs to generate questions with more skewed split ratios. The effect of this can be observed in the decrease in DP (and to a lesser dgree DC) performance when compared to the COMMON+ dataset of a similar size. Although the same can be observed for UoT and GoT, the decrease is less significant, possibly mitigated by the various forms of exploration and planning employed by both.

Method	Number of Items/Dataset		
	COMMON+	Skewed 128	Breeds
GPT 4.1 + Even split prompt			
GoT	9.4	9.2	6.4
UoT	10	10	7
DP	11.7	12.9	7.8
DC	12.3	12.6	9.0
GPT 4.1 + Natural prompt			
GoT	10.2	11.8	7.4
UoT	11	13	9
DP	13.8	16.2	7.8
DC	12.9	14.6	9.3
Qwen 2.5 72B Instruct + Even split prompt			
GoT	10.2	10.2	6.2
UoT	11	11	7
DP	11.9	14.6	8.0
DC	12.4	13.6	8.3
Qwen 2.5 72B Instruct + Natural prompt			
GoT	10.0	10.8	6.6
UoT	11	12	8
DP	12.7	19.2	8.0
DC	12.7	17.9	7.8

Table 1: **Worst case interaction length for various LLMs on SLSR**. We set $d = 3$.

SLSR with Synthetic Splits

The quality of a candidate question q in SLSR at some infoset $I(H)$ can be roughly measured using the ratio $|Y(S(H), q)| : |\bar{Y}(S(H), q)|$, representing how evenly q is able to split $S(H)$ into two. Given the difficulty in controlling the quality of LLM-sampled questions, we wish to study SLSR in a more controlled environment. Instead of using a LLM for question generation, we define a question generating function $g' : 2^{\mathcal{S}} \setminus \phi \times (0, 1) \to 2^{\mathcal{S}}$ which takes as input $S(H)$ and a split ratio $r \in (0, 1)$, and outputs the set[4] $Y' \subset S(H)$ (and consequently $\bar{Y}' = S(H) \setminus Y'$) at the fixed ratio $\frac{|Y'|}{|S(H)|} = r$. Two implementations of g' were considered: (i) Random Splits: Sample and return $r \cdot |S(H)|$ items uniformly at random from $S(H)$ without replacement. (ii) Feature Based Splits: For each $s \in S(H)$, generate k features $F_i = (f_1, ..., f_k)$, $f_j \in [0, 1]$ at the start of the game. g' randomly selects one of k features, and sorts $S(H)$ based on the selected feature. The top $r \cdot |S(H)|$ items are returned.[5]

The results are shown in Table 2. GoT consistently achieves superior performance compared to UoT, particularly as r decreases and the splits become increasingly skewed. This suggests that GoT provides the most improve-

[4]The question itself does not matter in this case, as both UoT and GoT is language agnostic.

[5]Random splits can be seen as Feature Based Splits with a very large k.

Method	r 0.4	0.33	0.25
3 Features			
GoT	9.4	10.0	13.8
UoT	10.0	11.0	15.0
5 Features			
GoT	9.2	10.2	13.2
UoT	10	11.0	15.0
Random Splits			
GoT	8.8	9.8	11.6
UoT	10	11	15

Table 2: **Worst case interaction length on SLSR with Synthetic splits with COMMON+**. d is set to be 3.

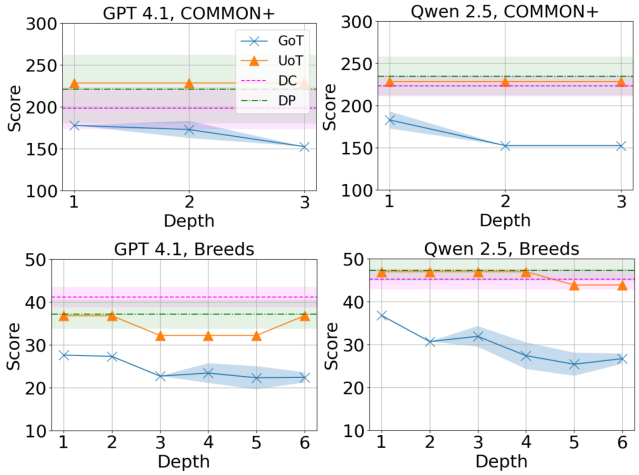

Figure 3: **Performance of various methods in WSLSR on COMMON+ and Breeds.** The x-axis is d, the y-axis is the payoff for the Answerer $w_i \cdot |H|$.

ment when the candidate questions are not optimal. This however does not translate well when actual questions are used due to other confounding factors not captured.

WSLSR

We further examine the performance in the WSLSR. Due to the change in payoffs in the full WSLSR game, we need to replace GoT's heuristic function used in the truncated games in a similar manner. The new heuristic function $h(l) = \max_{s \in S(l)} w(s) \cdot (d(l) + \log_2(|S(l)|))$ is formulated by multiplying the weight of the most important item with the estimated length of the game, thereby reflecting the structure of the payoff $w(s^*) * |H|$. UoT remains unchanged, as it was unclear how it should be appropriately modified. The question sampling prompt is modified to include the item weights and the payoff calculation. The item weights for each dataset is sampled from a lognormal distribution with parameters $\mu = 0$ and $\sigma = 1$.

The results can be seen in Figure 3. GoT outperforms UoT in most settings, with improvement ranging from 25% to 40%. We also noticed UoT being outperformed by DP and DC more frequently in WSLSR than in SLSR, possibly due

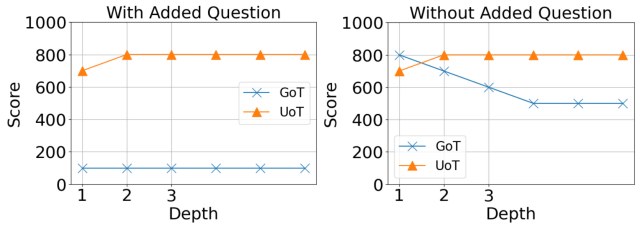

Figure 4: **Performance in WSLSR with artificially skewed weights on Breeds.** Graph on the left shows the performance when the additional question is included.

to UoT not accounting for the weight of items. This may lead to situations where questions yielding higher information gain (under the assumption of a uniform distribution) are preferred over those that more efficiently isolate heavily weighted items, a less optimal strategy in WSLSR.

We also examined the effect of increasing the simulation depth d to up to 6 using the smaller Breeds dataset. In general performance of GoT improves as d increases, but plateaus when it approaches what we hypothesize to be the theoretical best strategy for the given game.

A brief study on the effect of question quality in WSLSR was also conducted. An artificially skewed item weight distribution for the Breeds dataset is constructed by assigning all but one item with a weight of 1, and the remaining item being assigned a weight of 100. The optimal strategy in this case is to ask if the item with the heaviest weight is the correct item as the first question. We confirmed this by manually adding this question to the set of candidates at the start of the game and as seen in Figure 4, GoT consistently arrives at the aforementioned optimal strategy. However when this question is not manually added, the performance falls short of our expected performance by a large margin. Indicating that the quality of the questions can significantly affect the the performance of GoT.

7 Limitations and Future Work

We defined f to have a output of either 0 or 1, limiting the candidate questions to have strictly binary answers. How this can be extended to include questions with open ended responses is left as future work. We only experimented with three candidate questions at each infoset. The effect of more candidate questions was not studied due to time constraints.

8 Conclusion

In this study we formalized the definition of SLS and its variants as a way to quantitatively measure LLM's information-seeking abilities, and proposed GoT, a simple yet effective approach to play the SLSR game. We show that GoT is able to improve the worst case performance as compared to some prior methods. Our approach of optimizing for the worst case performance can often be more valuable than to optimize for the average performance in certain scenarios.

References

An, B.; Tambe, M.; and Sinha, A. 2017. Stackelberg security games (ssg) basics and application overview. *Improving Homeland Security Decisions*, 485.

Bertolazzi, L.; Mazzaccara, D.; Merlo, F.; and Bernardi, R. 2023. ChatGPT's Information Seeking Strategy: Insights from the 20-Questions Game. In Keet, C. M.; Lee, H.-Y.; and Zarrieß, S., eds., *Proceedings of the 16th International Natural Language Generation Conference*, 153–162. Prague, Czechia: Association for Computational Linguistics.

Brown, N.; and Sandholm, T. 2017. Safe and nested subgame solving for imperfect-information games. *Advances in neural information processing systems*, 30.

Brown, N.; and Sandholm, T. 2018. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374): 418–424.

Brown, N.; and Sandholm, T. 2019. Superhuman AI for multiplayer poker. *Science*, 365(6456): 885–890.

Burch, N.; Johanson, M.; and Bowling, M. 2014. Solving imperfect information games using decomposition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

(FAIR)†, M. F. A. R. D. T.; Bakhtin, A.; Brown, N.; Dinan, E.; Farina, G.; Flaherty, C.; Fried, D.; Goff, A.; Gray, J.; Hu, H.; et al. 2022. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science*, 378(6624): 1067–1074.

Goldman, S. A.; and Kearns, M. J. 1995. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1): 20–31.

Grand, G.; Pepe, V.; Andreas, J.; and Tenenbaum, J. B. 2024. Loose LIPS Sink Ships: Asking Questions in Battleship with Language-Informed Program Sampling. *ArXiv*, abs/2402.19471.

Handa, K.; Gal, Y.; Pavlick, E.; Goodman, N. D.; Andreas, J.; Tamkin, A.; and Li, B. Z. 2024. Bayesian Preference Elicitation with Language Models. *ArXiv*, abs/2403.05534.

Hebart, M.; Dickter, A.; Kidder, A.; Kwok, W.; Corriveau, A.; Wicklin, C.; and Baker, C. 2019. THINGS: A database of 1,854 object concepts and more than 26,000 naturalistic object images. *PLOS ONE*, 14: e0223792.

Hu, Z.; Liu, C.; Feng, X.; Zhao, Y.; Ng, S.-K.; Luu, A. T.; He, J.; Koh, P. W.; and Hooi, B. 2024. Uncertainty of thoughts: Uncertainty-aware planning enhances information seeking in LLMs. In *Proceedings of the 38th Conference on Neural Information Processing Systems*, NeurIPS '24. Red Hook, NY, USA: Curran Associates Inc. ISBN 9798331314385.

Jain, M.; An, B.; and Tambe, M. 2013. Security games applied to real-world: Research contributions and challenges. In *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, 15–39. Springer.

Kovarık, V.; Seitz, D.; and Lisy, V. 2021. Value functions for depth-limited solving in imperfect-information games. In *AAAI Reinforcement Learning in Games Workshop*, volume 132, 133–138.

Kuhn, L.; Gal, Y.; and Farquhar, S. 2022. CLAM: Selective Clarification for Ambiguous Questions with Generative Language Models.

Kwon, W.; Li, Z.; Zhuang, S.; Sheng, Y.; Zheng, L.; Yu, C. H.; Gonzalez, J. E.; Zhang, H.; and Stoica, I. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Lanctot, M.; Lockhart, E.; Lespiau, J.-B.; Zambaldi, V.; Upadhyay, S.; Pérolat, J.; Srinivasan, S.; Timbers, F.; Tuyls, K.; Omidshafiei, S.; et al. 2019. OpenSpiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*.

Li, B. Z.; Kim, B.; and Wang, Z. 2025. QuestBench: Can LLMs ask the right question to acquire information in reasoning tasks? *ArXiv*, abs/2503.22674.

Li, B. Z.; Tamkin, A.; Goodman, N. D.; and Andreas, J. 2023. Eliciting Human Preferences with Language Models. *ArXiv*, abs/2310.11589.

Li, S. S.; Balachandran, V.; Feng, S.; Ilgen, J. S.; Pierson, E.; Koh, P. W.; and Tsvetkov, Y. 2024. MediQ: Question-Asking LLMs and a Benchmark for Reliable Interactive Clinical Reasoning. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *Advances in Neural Information Processing Systems*, volume 37, 28858–28888. Curran Associates, Inc.

Liu, B.; Guertler, L.; Yu, S.; Liu, Z.; Qi, P.; Balcells, D.; Liu, M.; Tan, C.; Shi, W.; Lin, M.; et al. 2025. SPIRAL: Self-Play on Zero-Sum Games Incentivizes Reasoning via Multi-Agent Multi-Turn Reinforcement Learning. *arXiv preprint arXiv:2506.24119*.

Liu, M.; Farina, G.; and Ozdaglar, A. 2024. LiteEFG: An Efficient Python Library for Solving Extensive-form Games.

Moravčík, M.; Schmid, M.; Burch, N.; Lisỳ, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; and Bowling, M. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337): 508–513.

Moravcik, M.; Schmid, M.; Ha, K.; Hladik, M.; and Gaukrodger, S. 2016. Refining subgames in large imperfect information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

OpenAI. 2024. GPT-4.1. Https://openai.com/index/gpt-4-1/.

Perolat, J.; De Vylder, B.; Hennes, D.; Tarassov, E.; Strub, F.; de Boer, V.; Muller, P.; Connor, J. T.; Burch, N.; Anthony, T.; et al. 2022. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*, 378(6623): 990–996.

Piriyakulkij, W. T.; Kuleshov, V.; and Ellis, K. 2023. Active Preference Inference using Language Models and Probabilistic Reasoning. *ArXiv*, abs/2312.12009.

Pita, J.; Jain, M.; Ordónez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. ARMOR Security for Los Angeles International Airport. In *AAAI*, 1884–1885.

Shieh, E.; An, B.; Yang, R.; Tambe, M.; Baldwin, C.; DiRenzo, J.; Maule, B.; and Meyer, G. 2012. Protect: A deployed game theoretic system to protect the ports of the united states. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems-volume 1*, 13–20.

Shoham, Y.; and Leyton-Brown, K. 2008. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

Siegler, R. S. 1977. The Twenty Questions Game as a Form of Problem Solving. *Child Development*, 48(2): 395–403.

v. Neumann, J. 1928. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1): 295–320.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *nature*, 575(7782): 350–354.

Von Stengel, B. 1996. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2): 220–246.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E. H.; and Zhou, D. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *ArXiv*, abs/2203.11171.

Xu, J.; Wang, Y.; Tang, D.; Duan, N.; Yang, P.; Zeng, Q.; Zhou, M.; and Sun, X. 2019. Asking Clarification Questions in Knowledge-Based Question Answering. In Inui, K.; Jiang, J.; Ng, V.; and Wan, X., eds., *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 1618–1629. Hong Kong, China: Association for Computational Linguistics.

Yang, Q. A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Dong, G.; Wei, H.; Lin, H.; Yang, J.; Tu, J.; Zhang, J.; Yang, J.; Yang, J.; Zhou, J.; Lin, J.; Dang, K.; Lu, K.; Bao, K.; Yang, K.; Yu, L.; Li, M.; Xue, M.; Zhang, P.; Zhu, Q.; Men, R.; Lin, R.; Li, T.; Xia, T.; Ren, X.; Ren, X.; Fan, Y.; Su, Y.; Zhang, Y.-C.; Wan, Y.; Liu, Y.; Cui, Z.; Zhang, Z.; Qiu, Z.; Quan, S.; and Wang, Z. 2024. Qwen2.5 Technical Report. *ArXiv*, abs/2412.15115.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *ArXiv*, abs/2305.10601.

Zhang, B.; and Sandholm, T. 2021. Subgame solving without common knowledge. *Advances in Neural Information Processing Systems*, 34: 23993–24004.

Zhang, X.; Deng, Y.; Ren, Z.; Ng, S.-K.; and Chua, T.-S. 2024. Ask-before-Plan: Proactive Language Agents for Real-World Planning. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *Findings of the Association for Computational Linguistics: EMNLP 2024*, 10836–10863. Miami, Florida, USA: Association for Computational Linguistics.

Zhang, Y.; Lu, J.; and Jaitly, N. 2024. Probing the Multi-turn Planning Capabilities of LLMs via 20 Question Games. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1495–1516. Bangkok, Thailand: Association for Computational Linguistics.

Zinkevich, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2007. Regret minimization in games with incomplete information. *Advances in neural information processing systems*, 20.

A Strategy Representation

There are several ways to represent strategies (up to payoff equivalence) in imperfect information games with perfect recall. For computational reasons, the *sequence form* is often used . We briefly describe the other representations.

1. Normal/Strategic Form. The was what was presented in the paper. The set of deterministic strategies is the cartesian product of actions at each infoset $\Pi = \prod_{I \in \mathcal{I}} \mathcal{A}(I)$. Thus, every policy $\pi \in \Pi$ tells us exactly what action to take at every information set that could be encountered. Randomized strategies are distributions over deterministic policies, i.e., Δ_Π. The size of Π is exponential in the number of infosets.

2. Reduced Normal Form. The reduced normal form prunes the number of normal form strategies by grouping together strategically equivalent ones. See any textbook in game theory, or the work by Von Stengel (1996) for a more complete discussion. Let I be an infoset and $\alpha, \alpha' \in \mathcal{A}(I)$ be distinct actions in I. Let I' be an infoset that α' precedes. Then, normal form strategies that choose α in I have payoffs (regardless of opponent strategy) is independent of what action is taken in I', since to enter I' at all would require choosing α'. Thus, two strategies π, π' that choose α in I but differ in actions taken in I' are strategically equivalent. The minimal set of strategies that are obtained by forming such equivalence classes forms the set of reduced-normal form strategies, which can be significantly smaller than normal form ones. As before, strategies can be randomized in reduced normal form as well. Unfortunately, the size of reduced normal form strategies can still be exponential in the number of infosets.

3. Behavioral Form. The behavioral strategies involve placing a *distribution* of actions (possibly not deterministic) at each information set, thus the size is linear in the total number of actions summed over all infosets, which in turn is no larger than the size of the game tree. It can be shown using *Kuhn's theorem* that under perfect recall, the set of behavioral strategies are strategically equivalent to (reduced) normal form strategies. Unfortunately, while behavioral strategies are intuitive and compact, optimizing in behavioral form is difficult as payoffs are nonlinear in the strategy representation.

4. Sequence Form. The sequence form strategy alleviates the problems assigning probabilities to sequences σ (essentially actions when the game has perfect recall) of taking a particular sequence in isolation from chance and other players. At each infoset I, the sequence form is essentially identical to the behavioral form (a probability simplex) except that they are normalized by the probabilities given by the infoset's parent sequence $\sigma(I)$, which is the last action (sequence) taken before reaching I — this is guaranteed to be unique because of perfect recall. Thus, the sequence form is the same size as behavioral strategies (except for an extra "empty sequence" ϕ set to 1.0 that is the parent of all initial infosets. The sequence form strategy space is sometimes known as the treeplex

$$\mathcal{X} = \left\{ x \in \mathbb{R}_+^N \middle| x[\phi] = 1; \sum_{\sigma = Ia} x[\sigma] = x[\sigma(I)] \; \forall \, I \in \mathcal{I} \right\}$$

where $N = 1 + \sum_{I \in \mathcal{I}} |I|$ and $\mathcal{I}$ are the set of infosets, Ia refers to the sequences ending with action i starting at infoset I, and ϕ is the empty sequence. The vertices of Π are correspond to (reduced) normal form strategies. using the sequence form, we can write the Nash equilibrium of a zero-sum game as the bilinear saddle point problem

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} x^T M y$$

where M is the sequence form payoff matrix. Observe that the objective is linear in both x or y. This saddle point can be found efficiently using a variety of methods, including first order methods such as the counterfactual regret minimization (Zinkevich et al. 2007), linear programming or other first order methods (e.g., mirror prox). The library which we use (Liu, Farina, and Ozdaglar 2024) uses counterfactual regret minimization.

B Detailed Derivations

Proof of Theorem 1

This problem is equivalent to the following set-intersection problem.

Set intersection problem. Let $\mathcal{S}$ be a finite set and $\mathcal{K} \subseteq 2^{\mathcal{S}}$ a set of sets Given some fixed $s \in \mathcal{S}$, we want to find if there is a set $\mathcal{J} \subseteq \mathcal{K}$ such that $|\mathcal{J}| \leq k$ and $\bigcap_{J \in \mathcal{J}} J = \{s^*\}$, i.e., can we find a small subset of $K \subseteq \mathcal{K}$ such that its intersection contains exactly s^*. One can see that $\mathcal{K}$ contains the set of questions that can be asked about s^*. We can safely assume that $s^* \in K \in \mathcal{K}$.

Set cover. Let $\mathcal{A}$ and $\mathcal{B} \subseteq 2^{\mathcal{A}}$. The decision problem is whether one can find some set $B^* \subseteq \mathcal{B}$ of size k or smaller such that $\bigcup B^* = \mathcal{A}$.

NP-completeness. We show that a solution to the set intersection problem corresponds to a solution to set cover and vice versa. We define $\mathcal{A} = \mathcal{S} \setminus \{s^*\}$ and $\mathcal{B}$ contains subsets that are the complements of the elements of $\mathcal{K}$, $\mathcal{B} = \left\{ B_i = \overline{K_i} \mid K_i \in \mathcal{K} \right\}$. Take a set cover $B^* \subseteq \mathcal{B}$ of $\mathcal{A}$ and its corresponding set K^* in $\mathcal{K}$. Since B^* covers $\mathcal{A}$, i.e., $\bigcup B^* = \mathcal{A}$, its corresponding $K^* = \left\{ \overline{K_i} \mid B_i \in B^* \right\}$ has intersection $\bigcap_{i:B_i \in B^*} K_i = \{s^*\} \cup \bigcap_{B_i \in B^*} \overline{B_i} = \{s^*\}$. Similarly, any solution K^* to the set intersection problem corresponds to a solution to set cover, since $\bigcup_{i:K_i \in K^*} B_i = \bigcup_{i:K_i \in K^*} \overline{K_i} = \overline{\bigcap_{i:K_i \in K^*} K_i} = \overline{\{s^*\}} = \mathcal{A}$. Finally, any solution to the set intersection problem can also be checked in polynomial time.

Proof of Theorem 2

It is clear that the even-split strategy uses exactly k questions. Furthermore, every question yields exactly one bit of information, so at least k questions are needed. This shows that the even-split strategy is a NE (that this is a minimax

solution). To show that the uniform strategy is a maximin solution, simply apply symmetry. Let $y^* \in \Delta_n$ be some (potentially non-uniform) NE for the answerer. We know that the set of maximin solutions for a zero-sum matrix game forms a non-empty convex set. Take all permutations of y^*. By symmetry, they must all be NE as well. Let the average over all permutations be be $\overline{y^*}$, again by symmetry this is equal to the uniform distribution. But by the aforementioned convexity of the set of maximin solutions this is also a Nash equilibrium for the Answerer.

Discussion of Theorem 3

We give a very brief overview of subgame search and maxmargin resolving. The finer details and mathematical formulation is omitted. The main goal is to establish that GoT is implemented in the same way that maxmargin resolving is, which in turn implies safety.

Safe subgame search Subgame search (also known as subgame resolving or continual resolving) is a class of methods used to perform depth-limited search in a principled manner in imperfect information extensive form games. The idea behind subgame search is to play a *blueprint* strategy until the player enters a *subgame* (an imperfect information subgame is a forest of trees, closed under both the descendant relation and membership within augmented information sets for any player, (Burch, Johanson, and Bowling 2014)). The blueprint strategy is *typically* the solution to a coarse, abstract version of the full game. Once inside the subgame, the player *resolves* for a better solution for the subgame that was reached (and that subgame only). This is analogous to the perfect information case where one only performs search (or refinement) of a strategy in the state that was reached in actual play, since solving the original full game is intractable.

A safe subgame solving algorithm is one that is guaranteed to perform no worse than the blueprint strategy. Naive subgame solving attempts to solve the subgame by constructing a gadget game starting with a chance node which leads to all initial states in the subgame based on the probabilities (under the blueprint and the opponent strategy whens solving the blueprint) of reaching them. This however, neglects the fact that resolving just the subgame that was reached could entice the opponent to change their action, rendering these chance probabilities incorrect, leading to a worse performance than the blueprint.

Maxmargin resolving Maxmargin resolving (Moravcik et al. 2016) is one such approach to perform subgame solving in a more principled fashion. The idea is to augment the gadget game such that the *opponent* first chooses which infoset (of the opponent) it would like to begin from, after which, a chance node enforces the probabilities of reaching each of those states (belonging to the opponent's infoset) is, based on the blueprint strategy of the main player. Letting the opponent choose which infoset (note that we may have to add dummy infosets if the opponent does not move at the beginning of the subgame) they want to begin the subgame with ensures that the main player optimizes for the minimum *margin*. Here, the margin for each of the opponent's infoset is the difference between the value in the infoset under the blueprint strategy versus the refined strategy. If the main player optimizes the value of a particular head infoset (of the opponent) too much at the expense of performing poorly at other infosets, then the opponent would choose those infosets instead, resulting in unsafe resolving. By allowing the opponent to choose initial infosets, Maxmargin avoids this explicitly by ensuring that all of the infosets are equally improved (for the main player) after refinement.

In practice, we do not actually know the values of initial states of the subgame under the best-response of the opponent is (without expanding the entire subgame to its leaves) and have to resort to approximate value functions. Note that it is not immediately clear what constitutes a good value function, since the value of a state will depend on play from both players and is complicated by imperfect information. See Kovarık, Seitz, and Lisy (2021) for a more thorough discussion.

GoT as maxmargin resolving First, observe that SLSR is a relatively simple EFG in that the Answerer only moves once at the beginning, after which there are no further interactions from it. In fact, we could allow the Answerer to have perfect information. Furthermore, the proper subgames (i.e., not the full game) are simply the histories H, or equivalently, the subgame beginning at $I(H)$, which comprise all questions asked but not what s^* was chosen. Note that each proper subgame only has the Questioner taking actions. Since the Answerer has perfect information the "head infosets" of a subgame $I(H)$ for the Answerer simply corresponds to the initial states of that history H. Note that we don't actually need to explicitly construct these head infosets: since the Answerer has full information, these are "dummy" infosets have essentially one action only, which lead to the corresponding state in the lead infoset of the Questioner. This shows that the structure of the gadget game of maxmargin is exactly the same as what we propose. An example of this construction is shown in Figure 5.

In maxmargin, the payoffs under each head infoset of the subgame is shifted by the payoffs under the blueprint (or function approximation). This ensures that the *margin* is optimized as opposed to absolute payoffs. Thankfully, no such shift is necessary in our case because the approximated values at each state is $\log_2(|S(H)|)$, i.e., the log of the number of items remaining. This value is the same for *every* state in the infoset $I(H)$, thus the value of the shift is the same over the entire subgame, which, as far as solving the subgame goes has no bearing on the refined strategy.

This same argument also extends to our choice of value function for WSLSR, $h(l) = \max_{s \in S(l)} w(s) \cdot (d(l) + \log_2(|S(l)|))$, which depends only on the set of items remaining, $S(H)$. However, if our value function was chosen to also depend on the Answerer's precise choice of s^*, then such an argument would no longer hold and we would have to perform these shifts in payoffs accordingly.

Remark. *In theory, maxmargin resolving only performs resolving upon entering a subgame (e.g., after $d = 3$ questions are asked). In practice, GoT performs resolving at every infoset that is encountered during actual play.*

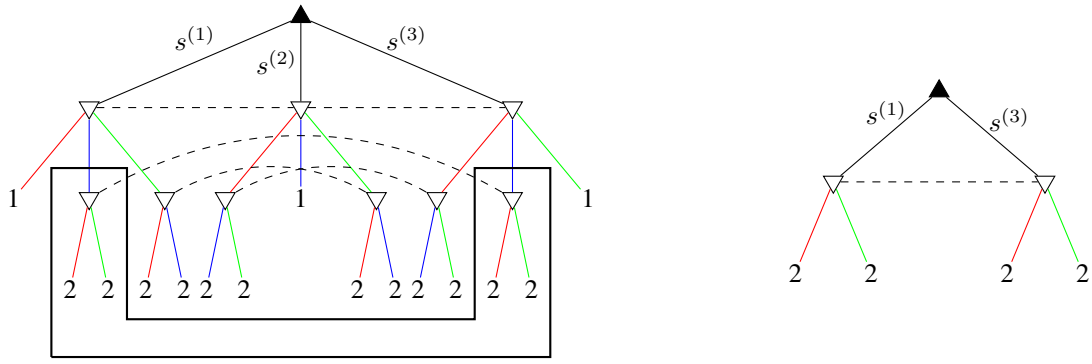

Figure 5: Left: example of a subgame based on Example 1 and Figure 1. Bounded region by thick dark lines form an example of a subgame corresponding to the history of asking $q^{(2)}$ and getting an answer of 1. Right: the gadget game used for solving the subgame shown on the left. Note that it is the Answerer who takes the first action, not chance (which would be the case for naive subgame solving), and the actions only include the items that are consistent with H, in this case $s^{(1)}$ and $s^{(3)}$.

C Implementation Details

General Implementation

Method Hyperparameters Since UoT shares a similar simulation procedure with GoT, we use the same values for the simulation depth d and the number of candidate questions m for both methods within each experiment, although these values may vary across different experiments. For DC, we similarly provide a set of m candidate questions which it may choose from.

Question Generation When There Are Two Items Left In instances during a (W)SLSR game where the current history H satisfies $|S(H)| = 2$, we deterministically construct two candidate questions of the form "Is x the correct item?" for each remaining item, instead of sampling questions from the LLM. While not strictly required, it is implemented primarily for efficiency. Under Assumption 6, any question that satisfies the assumption will successfully distinguish between the two remaining items, thereby terminating the game. This eliminates the need to query an LLM at such states, thereby reducing the latency associated with constructing the simulation tree.

Question Caching and Reuse During each execution of GoT, the set of candidate questions $g(S(H))$ generated by the LLM for any explored history H is cached for reuse. In the case of UoT and DC, if the current history has previously been explored by GoT, the corresponding cached set of candidate questions is reused in place of sampling new questions from the LLM. This is done to ensure a fair comparison across methods, as variations in the quality of candidate questions can substantially impact performance.

Handling erroneous LLM outputs For ease of parsing, we require the LLMs to produce outputs in JSON format. While the model occasionally fails to adhere to this format, we address such cases by retrying the generation. In our experiments, this retry mechanism proved effective, and no further issues were observed.

GoT Specific

When Simulation Tree is Terminal During Simulation Step (1), after constructing the local simulation tree, we verify whether each leaf node corresponds to a terminal state, i.e., whether $|S(l)| = 1$ holds for all leaf nodes. If this condition is met, the strategy derived from the current subgame is adopted for all subsequent question selections until the end of the game, rather than being used solely for the next question followed by resolving a new subgame at the next step. This is justified by the fact that the current subgame extends to the end of the game, thereby obviating the need for iterative strategy construction in subsequent steps.

D Additional Results

Item Weights

Figure 6 presents a histogram of item weights for the Breeds and COMMON+ datasets used in WSLSR. The weights were assigned randomly, as there was no clear method for determining meaningful valuations for each item, given the nature of the datasets.

It is important to note that the performance of GoT in a WSLSR game may be sensitive to the underlying distribution of item weights. In particular, when the weights were sampled from a uniform distribution, we observed that the relative performance of GoT compared to UoT mirrored the results in the standard SLSR setting, providing a consistent but smaller improvement.

Randomization of GoT's strategies

We quantified the randomness of GoT's strategy using entropy, as shown in Figure 7. In general, the resulting strategies exhibit higher entropy in the standard SLSR game compared to the WSLSR variant. This is likely due to the item weight distribution being tail-heavy, which encourages the strategy to favor candidate questions that more efficiently isolate the heavily weighted items. Nonetheless the strategies remain sufficiently random, suggesting that we do not encounter degenerate cases in which a deterministic strategy would be optimal.

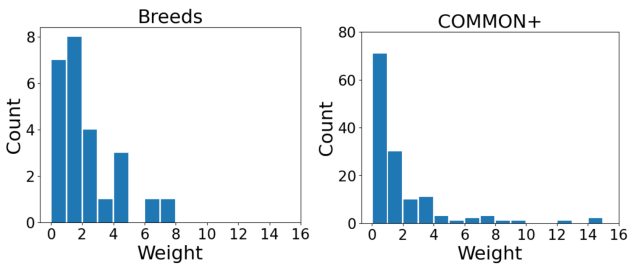

Figure 6: **Distribution of item weights used in our experiments.** Weights are sampled from a lognormal distribution with parameters $\mu = 0, \sigma = 1$.

Full SLSR Games

For each SLSR game, there exists a lower bound on the expected worst case number of questions required. This bound can be obtained by solving the fully specified game. As this is only feasible for smaller datasets, we constructed complete SLSR games on the smaller Breeds dataset and experimented on GoT and UoT by setting the simulation depth d to game tree depth D. Since constructing the full game tree is required, we enforce Assumption 8 to ensure that all branches eventually terminate. The results are shown in table 3, where GoT's performance aligns with the lower bound on the performance. On the other hand, UoT's performance showed minimal improvement even when provided with the full game tree during simulation, suggesting that achieving the lower bound requires a non-deterministic strategy.

Method	Experiment	
	Subgame ($d = 3$)	**Full Game** ($d = D$)
GPT 4.1 + Even split prompt		
GoT	6.4	5.84
UoT	7	7
GPT 4.1 + Natural prompt		
GoT	7.4	5.64
UoT	9	8
Qwen 72B Instruct + Even split prompt		
GoT	6.2	5.44
UoT	7	7
Qwen 72B Instruct + Natural prompt		
GoT	6.6	5.88
UoT	8	8

Table 3: **Worst case interaction length for on full SLSR Game.** Dataset used is the Breeds Dataset.

E Discussion on Assumptions

As noted, Assumptions 1 to 3 are inherent in the formal definition of the game. The remaining assumptions are either naturally satisfied by the experimental setup or are explicitly enforced during the execution of our method.

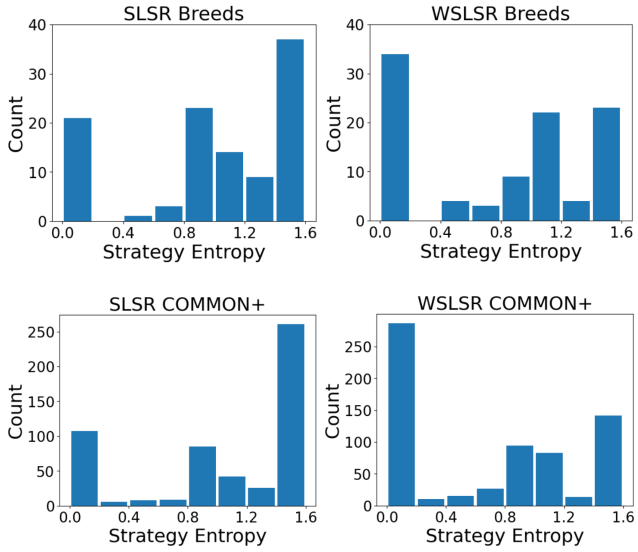

Figure 7: **Entropy of the strategies obtained using GoT at each infoset.** As $m = 3$, the maximum entropy of the strategy is $\log_2(3) \approx 1.58$ when every candidate question is chosen with equal probability. Strategies for when there are only two items left are not included.

Assumption 4 We employed LLMs, accessed both via online APIs and locally hosted instances, to implement the function f. For online API-based queries, the average latency per query is approximately 1 second, while for locally hosted models, the latency is slightly lower, typically falling just below 1 second.

Assumption 5 This assumption naturally follows from Assumption 6 in the (W)SLSR version of the game.

Assumption 6 This assumption holds in nearly all cases when LLMs are used as the question generator g. To enforce this, we verify whether there exists at least one question that permits progression of the game; if no such question is found, we resample the candidate questions. For certain experiments (such as full SLSR game solving) we impose a stronger version of this assumption:

Assumption 8. *For every subset $S \subseteq \mathcal{S}$, for every $q \in g(S)$, there exists a pair of distinct items $s, s' \in S$ where $f(q, s) \neq f(q, s')$*

This guarantees that the game tree will be finite, thereby enabling the complete construction of the game tree. To enforce this condition, we explicitly remove any candidate question that fails to satisfy the assumption from the set of candidate questions.

Assumption 7 This assumption is made to mitigate potential inaccuracies arising from LLM-based classifications. Although LLMs may produce erroneous outputs, handling such errors is beyond the scope of our method and is therefore not the primary focus of this work. Without this as-

sumption, classification errors during the simulation process could, in the worst case, require a restart of the game. Specifically, if the target item s^* is misclassified, it may be incorrectly eliminated from the set of possible items upon receiving a contradictory response from the answerer. Moreover, we observed that such misclassifications and consequently restarts occurred frequently when attempting to reproduce the results of (Hu et al. 2024), which drastically affected the worst case performance. Since this does not influence the Questioner's strategy under any of the evaluated methods, we believe it does not compromise the fairness of the experimental results. To enforce this assumption for both GoT and UoT, we depart from the approach used in (Hu et al. 2024), where a separate LLM instance was employed to simulate the Answerer. Instead, we cache the item set splits generated during the simulation step corresponding to each question and reuse them as the Answerer's response. This strategy ensures consistency between the item classifications performed during the simulation phase and the responses observed during actual execution of the (W)SLSR game.

F Dataset Details

The all constructed datasets (along with the weights for Breeds and COMMON+ used for WSLSR) are released alongside the accompanying code. To construct the datasets, we relied on publicly available sources. For example, the American Kennel Club for the Breeds dataset, and ChatGPT for the Skewed 128 dataset. All AI-generated datasets were subsequently verified to ensure that each item included corresponds to a real-world entity.

G Prompts Used

The prompts used for sampling questions (to implement the function g) for the SLSR and WSLSR games are shown in tables 4 and 5 respectively. DP uses the same prompts for asking questions by setting $m = 1$. DC additionally uses the prompt in table 6 to choose the question from the candidate set. The prompts used to generate answers to questions for items (to implement the function f) is shown in table 7.

H Examples of Questions Asked

Unnatural Questions A typical question posed in a (W)SLSR game, using the COMMON+ dataset, resembles the following:

- Is the item something a person can physically pick up with their hands (assuming average human size and strength)?
- Is the item man-made?
- Is the item an animal?

Such questions generally pertain to intrinsic properties of the items themselves.

In contrast, when the even prompt was used, we occasionally observed the generation of more "unnatural" questions:

- Is the breed's name made up of only one word?
- Is the name of the item longer than 7 characters?
- Does the name of the item contain the letter 'a'?
- Is the item's name longer than four letters?
- Does the item's name start with a letter from A-M (inclusive)?

These questions typically reference the item names directly, rather than the underlying items themselves, and can be interpreted as the LLM's attempt to optimize the question for the SLSR game. These occurrences were primarily observed when GPT-4.1 was used as the LLM, we did not observe this when using Qwen 2.5.

Ambiguous Questions In our formulation of the SLS problem, we assume that (i) the codomain of f is binary, and (ii) each item $s \in \mathcal{S}$ has a unique, well-defined answer under f, as stated in Assumption 1. However, this assumption may not hold in real-world scenarios due to the inherent ambiguities and interpretive variability commonly present in natural language. One source of such ambiguity arises when an item name refers to multiple distinct entities. For example, consider the question: Is the Titanic something a person can pick up with their hands? The answer would be no if referring to the actual ship, but yes if referring to a DVD of the movie Titanic. While language ambiguity is a well-studied phenomenon in the field of Natural Language Processing, addressing such ambiguities remains challenging when applied to the more formal and rigorous framework of Game Theory. We believe that addressing this challenge presents a promising direction for future research.

I Experiment Hardware

Running the EFG solver was done on a AMD Ryzen 5 5500 CPU with 32 GB ram. Hosting the local instance of Qwen 2.5 72B Instruct LLM was done via the vLLM(Kwon et al. 2023) library on either 2 Nvidia H200 GPUs each with 141 GB memory, or 4 Nvidia H100 GPUs each with 80 GB memory.

Even Prompt

We are playing the game of ask and seek, where I am the answerer and you are the questioner.
I have chosen one item from a list of items, and your goal is to identify the correct item from a list of items by asking a series of yes or no questions.
Your should try to ask as few questions as possible.

Here are all the items:
[items]

One of the items is the correct item, please design questions about those items that helps you find the right item and can only be answer by Yes or No.
Notably, this question should fulfill that the count of YES items and NO items are almost the same with a permissible discrepancy of no more than one!
You should respond with the best **[m]** questions you can think of.

Please follow the JSON template:

```
{
    "questions" : [
        {
            "question" : "<question>"
            "reason" : "<reason>"
        },
        ...
    ]
}
```

Natural Prompt

We are playing the game of ask and seek, where I am the answerer and you are the questioner.
I have chosen one item from a list of items, and your goal is to identify the correct item from a list of items by asking a series of yes or no questions.
Your should try to ask as few questions as possible.

Here are all the items:
[items]

One of the items is the correct item, please design questions about those items that helps you find the right item and can only be answer by Yes or No.
You should avoid asking unnatural questions such as ones about the number of words the item has, or any such similar questions.
You should respond with the best **[m]** questions you can think of.

Please follow the JSON template:

```
{
    "questions" : [
        {
            "question" : "<question>"
            "reason" : "<reason>"
        },
        ...
    ]
}
```

Table 4: **Prompts used to sample candidate questions for SLSR.** The main difference between the two prompts are *italicized*. The items are represented as a python list, e.g. [‘Oppenheimer’, ‘Alan Turing’, ‘A Beautiful Mind’], while m is an integer

Weighted Prompt

We are playing the game of ask and seek, where I am the answerer and you are the questioner.
I have chosen one item from a list of items, and your goal is to identify the correct item from a list of items by asking a series of yes or no questions.
Each item has a value associated with it, and your penalty score will be the number of questions asked until you find the right item multiplied by the value of the correct item. You should try to minimize this score.

Here are all the items and their values: **[items with weights]**

One of the items is the correct item, please design questions about those items that helps you find the right item and can only be answer by Yes or No.
You should avoid asking unnatural questions such as ones about the number of words the item has, or any such similar questions.
You should respond with the best **[m]** questions you can think of.

Please follow the JSON template:

```
{
    "questions" : [
        {
            "question" : "<question>"
            "reason" : "<reason>"
        },
        ...
    ]
}
```

Table 5: **Prompt used to sample candidate questions for WSLSR.** Items with weights are represented as a python dictionary, e.g. {'Oppenheimer' : 3, 'Alan Turing' : 2, 'A Beautiful Mind' : 2}, while m is an integer

DC Choosing Prompt for SLSR

We are playing the game of ask and seek, where I am the answerer and you are the questioner.
I have chosen one item from a list of items, and your goal is to identify the correct item from a list of items by asking a series of yes or no questions.
Your should try to ask as few questions as possible.

Here are all the items:
[items with weights]

Here are some Yes or No question about them:
[question]

One of the items is the correct item, please choose one question from the list of questions that helps you find the right item.

Please follow the JSON template:

```
{
    "quesiton_choice" : int
    "reason" : "<reason>"
}
```

DC Choosing Prompt for WSLSR

We are playing the game of ask and seek, where I am the answerer and you are the questioner.
I have chosen one item from a list of items, and your goal is to identify the correct item from a list of items by asking a series of yes or no questions.
Each item has a value associated with it, and your penalty score will be the number of questions asked until you find the right item multiplied by the value of the correct item. You should try to minimize this score.

Here are all the items:
[items with weights]

Here are some Yes or No question about them:
[question]

One of the items is the correct item, please choose one question from the list of questions that helps you find the right item.

Please follow the JSON template:

```
{
    "quesiton_choice" : int
    "reason" : "<reason>"
}
```

Table 6: **Prompts used by DC to choose questions.** Questions are represented as a python dictionary, e.g. {0 : 'Related to codes?', 1 : 'Is it a movie?', 2 : 'Is it a person?'}

Response Prompt

Here are some items:
[items]

Here is a Yes or No question about them:
[question]

Please classify the items above based on this question. You should ensure that each item should only appear once in the final classification. If you are unsure about any item, you can mention it in the elaboration.

Please follow the JSON template:

```
{{
    "yes" : [
        "<yes_item>", ...
    ],
    "no" : [
        "<no_item>", ...
    ],
    "elaboration" : "<elaboration>"
}}
```

Table 7: **Prompt used to get answers to questions.** This is used for both SLSR and WSLSR.