Evaluation Section for ARLO

In this section, we evaluate ARLO's scalability (The growth in time to make architectural decisions as a function of the number of requirements) and sensitivity (How the decisions change as a function of changes in requirements) characteristics. Other characteristics, such as justifiability (The clarity in how decisions are made and the ability to understand and trust these decisions) and learnability (The improvement in decision-making quality with accumulated experience or data), would also be appropriate for evaluation, the first two are more important to study since we can use them to decide if ARLO can be useful in real-world cases. In future work, we will consider conducting surveys and studies to evaluate them.

Evaluating ARLO's Scalability

In this section, we evaluate ARLO's scalability, defined as the growth in time needed to make architectural decisions as a function of the number of requirements. This is an important characteristic to evaluate since we want to know how ARLO would behave for software with many requirements.

Given that ARLO uses several steps to reach its decision, we evaluate each step separately and then combine them to reach an overall decision. Also, we consider a scenario in which requirements are provided iteratively rather than all at once, which resembles how software requirements are obtained in the real world, specifically under agile development processes.

A. Analyzing the scalability of ARLO's step 2, forming condition groups:

Algorithm 1 was used to identify condition groups. It has two loops: the outer loop reviews all requirements with conditions, and the inner loop reviews the existing groups to see which can contain each requirement.

1. In the worst case, each requirement is an ASR and has a condition, and all conditions are mutually exclusive. This means that the inner loop is Algo 1 has to repeat equal to the number of requirements added before it. So the total will be 0+1+n-1 = n(n-1)/2, or the complexity of the Algorithm is $O(n^2)$, where n is the number of NLRE.
2. In the normal case, we assume 15% of requirements are ASR, and 15% contain a condition. This reduces the time complexity to $kO(n^2)$, K ~ 0.0225.

The LLM call to evaluate the condition takes about 1 second (for 20 words per requirement). Based on what I found on the OpenAI site:

- OpenAI GPT-3.5-turbo: 73ms per generated token
- Azure GPT-3.5-turbo: 34ms per generated token
- OpenAI GPT-4: 196ms per generated token

- OpenAI GPT-4o: 50ms per generated token

We can consider three systems to see how long it would take to run the Algorithm:

System Size	Group Fitting Check Count	Normal Case	Worst Case
Small (Trello) ~ 100 Rs	4,950	10 Sec	8 Min
Medium (Shopify) ~ 1000 Rs	499,500	16 Min	13 hours
Large (Salesforce) ~ 2000 Rs	1,999,000	1 Hour	55 hours
X Large (SAP) ~ 5000 Rs	12,497,500	7 hours	14 days

B. Analyzing the scalability of ARLO's step 3, making optimal choices:

The evaluation results depend on the experiment settings around two key assumptions on how the decision points are related and the architect's goals:

1. **Decision Point Dependency:** Whether the choices made within one group can affect other groups.
2. **Architect's Goal:** If we are maximizing the entire weighted score vs having non-linear constraints, such as if a QA has a weight higher than 0, it must be selected

Therefore, we would consider our evaluation in the following two settings:

Setting	Decision Choice Dependency	Architect's Goal Linearity
S1: Assuming both of these	Independent	Linear
S2: Assuming at least one of these	Dependent	Non-Linear

We first provide a mathematical model of ARLO and use it to show how a specific setting can impact ARLO's results, which helps with conducting the evaluations.

Lema: *If choice groups are independent and the object function is linear (S1), the optimization problem reduces to several intra-group optimization problems that could be solved independently.*

Lema's Proof:

Given:

- A matrix where each row represents a choice and each column represents a QA, with values {+1,−1,0}.
- Weights associated with each quality, {w1,w2,...,wn}, where n is the number of QA.

The objective is to maximize the weighted sum of these values for each choice within a group. The weighted sum for a choice i is given by:

$$S_i = \sum_{j=1}^{n} w_j \cdot M_{ij} \quad (1)$$

A change in the value of an element Mij could result in a change in the selected choice if the change alters the relative order of scores such that a different choice, k, now has a higher score than the previously selected choice, or vice versa.

Mathematically, let's assume Mij is altered by ΔMij, resulting in a new score:

$S'_i = S_i + w_j \cdot \Delta M_{ij}$ (2)

For a change in selection from choice i to choice k, it must be true that $S_k' > S_i'$ when previously $S_i \geq S_k$. Therefore, the selection change occurs if the adjustment leads to

$(S_k - S_i) > w_j \cdot \Delta M_{ij}$ (3)

S1 reduces the optimization to several independent optimizations (one per group). Therefore, we can make the following assertions on how ARLO would scale:

- Adding more groups would have a linear impact on the scale since we need to solve a linear multiple of optimization problems.
- Adding more choices for groups: While the optimization problem regarding the number of choices has exponential order, this is less significant since we only expect a handful of choices per group.

With S2, we cannot divide the initial problem into multiple optimization problems. The optimization problem is, in general, NP. The behavior of ARLO depends on the following factors and could be calculated given parameters' values by using simulation methods such as Monte Carlo analysis:

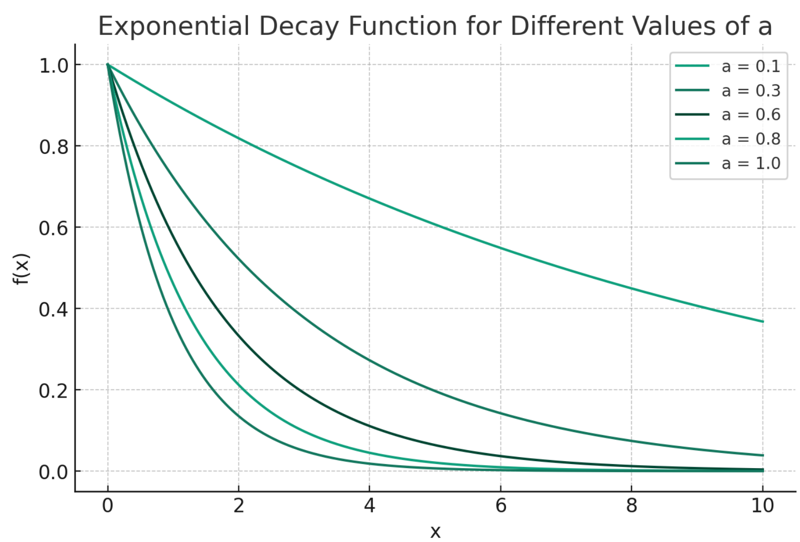
1. Matrix values
2. QA weights
3. Dependency among choices
4. The definition of the optimization goal

C. Considering iterative software development processes.

In iterative development, such as agile, the project is broken down into multiple sprints, and requirements are specified and provided to the development team over time rather than all simultaneously. We analyze the scalability of ARLO in such an interactive process. We use the following parameters in our analysis:

- n_i requirements are provided in sprint i.
- a_i ($0 < a_i < 1$) denotes the ratio of conditional ASR requirements.

For each new requirement in sprint i, we need to check them against the nominal condition group we had formed up until sprint i-1 and groups created in the current sprint based on conditional ASRs visited so far. In the worst case, we get a new group for each requirement ($a_i = 1$ for all i). In the average case, we can assume a_i starts from a number such as 20% and shrinks (since as we get more conditions, the chance of seeing a new condition reduces). So, we can model a_i as an exponential decay: $a_i = k.\exp(-ai)$, where a shows how fast the function reduces. For example, choosing a= 0.1 makes this almost linear for the first 10 iterations.



Let's make some assumptions to get to a mathematical model for the number of loop iterations for sprint i (S_i) and S for the total number of loop iterations.

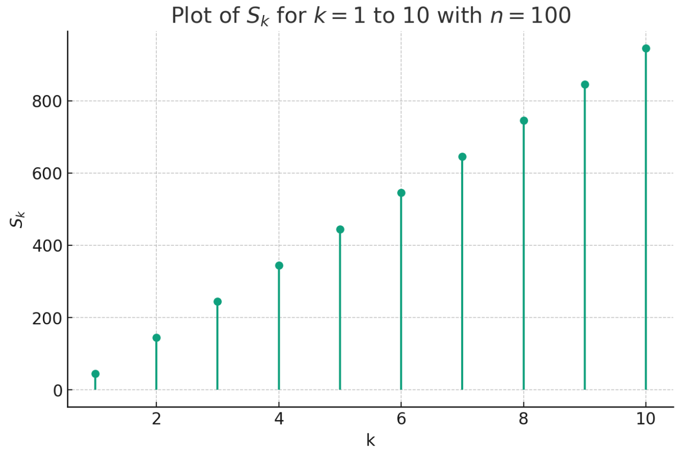
1. The project has 10 sprints.
2. Same number of requirements over 10 sprints, so $n_i = 0.1n$.

Worst Case Calculations:

- $a_i = 1$
- Sprint 1: $S_1 = 0 + 1 + \ldots + 0.1n - 1 = 0.05n(0.1n - 1)$
- Sprint 2: $S_2 = 0.1n + 0.1n + 1 + \ldots + 0.2n-1 = 0.1n(0.2n - 1) - 0.05n(0.1n - 1) = 0.05n(0.4n-2-0.1n+1) = 0.05n(0.3n-1)$
- …
- Sprint 10: $S_{10} = 0.9n + 0.9n + 1 \ldots + n - 1 = 0.05n \cdot (1.9n-1)$
- Or in general $S_i = 0.05n[0.1(2k-1)n-1]$

- The total number of iterations is the same as just having one iteration, S = 0.5n(n-1).

The plot for S_k is shown below:



Average Case Calculations:

- Set the ratio of new condition groups at iteration i to a_i = 0.2exp(0.1(i-1)).
- Sprint 1, S_1 = 0 + 1 + … + 0.02n = 0.01n(0.02n-1)
- S_i = [0.02n.exp(0.1(i-1))][0.02n.exp(0.1(i-1)) -1]/2
- We cannot come up with a closed form for S, but we can numerically calculate S_i for different numbers of requirements shown below:

n	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Total	Time
100	1	1	2	2	3	4	5	6	8	10	41	4 Sec
1,000	190	233	286	351	430	527	646	791	968	1,185	5,608	9 Min
2,000	780	955	1,169	1,431	1,751	2,142	2,620	3,204	3,918	4,791	22,759	37 Min
5,000	4,950	6,052	7,398	9,043	11,053	13,509	16,509	20,175	24,654	30,125	143,469	4 Hours

This shows that iterative development can reduce the processing time by helping (assuming we get fewer condition groups in later iterations).

Evaluating ARLO's Sensitivity

In this section, we evaluate ARLO's sensitivity, defined as how decisions change as a function of changes in requirements. This is an important characteristic to evaluate since we want to know how ARLO would behave for software when some requirements change or new requirements are added, which frequently happens in real-world software projects.

The sensitivity to condition groups can be evaluated by considering cases where a condition is misclassified, impacting the weight of one or more QAs.

With S1, the optimization is applied to each choice group independently. So, within that group, we can expect changes in decision-making if the QA's weight times the impact of another choice becomes favorable compared to the choice that would have been selected without a mistake. The following cases could happen:

1. The new QA has a positive (+1) impact on the choice that was correctly chosen. In this case, we won't observe a change in our choices.
2. The new QA has the same or worse impact on other choices than the correctly chosen one. In this case, we won't observe a change in our choices.
3. The new QA has a worse impact on the correct choice than other choices. In this case, we might observe a change. It depends on the weight of the desired QAs within the condition group. Hence, The degree of sensitivity of ARLO depends on the matrix values and weights associated with the desired and wrongly associated QAs.

Similar to what we mentioned for scalability analysis, with S2, we cannot divide the initial problem into multiple optimization problems, one for each choice group. The optimization problem is, in general, an NP problem.