Meaningful Human-in-the-Loop Checking of GenAI Synthesis for Restricted Languages

Siddhartha Prasad
Brown University

Skyler Austen
Brown University

Kathi Fisler
Brown University

Shriram Krishnamurthi
Brown University

Abstract

Developers routinely use GenAI tools (large language models enriched in various ways) to generate useful components of programs, such as regular expressions. While pleasant and often effective, this can easily lead to subtle bugs. The developer may have been unclear in their specification, they may not fully understand the language of the output, there may be systematic misconceptions suffered by the user and perhaps even embedded in the language model, and so on.

Responsible use of GenAI requires humans in the loop. To be effective, the human interaction must be both meaningful and moderate. We accomplish this as follows. First, we generate multiple candidate expressions instead of one. We then use formal language containment properties to generate distinguishing concrete scenarios that illustrate the *differences* between the candidates. We then have users rate these concrete scenarios. This process converges in a few steps, while also giving the user insight into any lack of clarity on their part.

We have built a tool, PICK, that implements this iterative process. We apply it to three formal languages with the necessary properties: regexes, linear temporal logic, and access-control policies. We show through experiments that PICK is a significant improvement over showing users the candidate expressions, and also helps catch situations where no output is a match.

2012 ACM Subject Classification Software and its engineering → Software notations and tools; Computing methodologies → Artificial intelligence; Theory of computation → Formal languages and automata theory

Keywords and phrases Regex, LTL, Access Control, Generative AI, Human-in-the-Loop

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2026.7

Supplementary Material *Software (VS Code extension)*: `https://zenodo.org/records/19631605`
Dataset (User-study materials): `https://zenodo.org/records/19631605`

Funding Partially supported by US NSF grants 2227863 and 2433429.

Acknowledgements We thank Tim Nelson, Rob Lewis, Will Crichton, Elijah Rivera, Gavin Gray, Nikos Vasilakis, Sam Tobin-Hochstadt, Chung-chieh Shan, Nishka Desai, Ariel Hirschhorn, and Alyxandra Harp Rose for helpful discussions; Ben Motz for sharing his seminar materials; and Rob Goldstone for cheerfully putting up with our half-formed ideas in many useful conversations. We appreciate the helpful reviewer feedback.

1 A Promise and a Pitfall

Consider a roboticist who needs to turn an English description into an LTL formula, an administrator who has to turn one into an access-control policy, or a developer who needs to generate a regex. These are all tasks for which they might turn to Gen(erative)AI. Suppose

they ask for a regex to check whether a dd-mm-yyyy string represents a valid date. They might get a response like:[1]

```
^(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])-(\d{4})$
```

along with a warning that this checks only structure but not whether specific day-month-year combinations are valid. Following up to request a regex that includes date validity produces:

```
^(?:(?:31-(?:0[13578]|1[02])-(?:\d{4}))|
   (?:29|30-(?:0[13-9]|1[0-2])-(?:\d{4}))|
   (?:29-02-(?:(?:\d\d(?:0[48]|[2468][048]|[13579][26]))|
             (?:[048]000)))|
   (?:0[1-9]|1\d|2[0-8]-(?:0[1-9]|1[0-2])-(?:\d{4})))
```

Given the complexity of the generated regex, the developer is unlikely to be able to inspect it for correctness. Even if they did, they might hold *misconceptions* about how regexes [45, 55, 59] (or likewise for other notations like LTL [11, 23, 26]) work; these same misconceptions may also be embedded in a language model used by the GenAI. They could write tests, but will they be comprehensive? And will they catch the edge cases needed to validate a complex output they do not entirely understand?

This might just seem like a matter of "LLM quality". However, even given a "perfect" LLM, there are many situations that call for human judgment. Suppose, for instance, we are generating a regex to categorize countries of "North America". Exactly which countries are in and out? What about the Caribbean? How about Caribbean countries below the Equator? The exact list of the world's countries is a fraught political matter. The list of honorific titles varies heavily by country. These are not a matter of "correctness": these questions have no canonical answers. Generating formal outputs from prose must contend with not only ambiguous prose but also intent, regional variation, contested world knowledge, and more.

Thus, while GenAIs are powerful aids for developing formal statements and several tools use them [10, 12, 18, 19, 39, 44, 50, 61, 79, 80, 83], we must use them responsibly, taking into account the above weaknesses and ambiguities. While numerous people have sounded the alarm about over-reliance, virtually none of the tools we have cited engage with humans in any meaningful way, often using GenAI to check the output—which is of no use in case of an erroneous specification, systematic misconception, etc. (Section 6 discusses this in more detail.) Safety, privacy, personal and political sensitivity, and so on all demand caution.

We instead take the philosophical position that, instead of just piling GenAI on top of GenAI, humans *must* be in the loop. However, the demands of humans must necessarily be:

Meaningful Asking humans to pass judgment on complex and abstract statements, such as those shown earlier, is unlikely to be effective. Laziness, automation bias, inability to form good judgments, and a desire to get things done will all lead to meaningless confirmation, just as we have seen for security and medical alerts [43, 70, 72].

Moderate Asking lots of questions, no matter how simple, can be exhausting and will also lead to errors as the number of questions grows. We should try to make every human action be highly impactful and not ask users to perform too many actions.

We embody this stance in a tool-supported workflow, shown in Figure 1 (Section 2 provides details), called PICK (short for Pairwise Iterative-Choice Knockout). It *uses concrete examples to guide a user in choosing between plausible formalized versions of a prompt*:

[1] These were generated by GPT-5 on 2025-09-15 via ChatGPT.com.

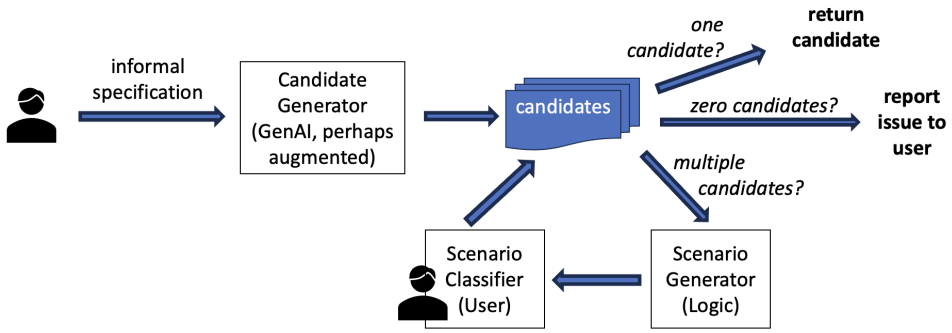


Figure 1 The Workflow of PICK.

1. A user provides a textual description of the desired formal expression (e.g., a regex). PICK supports any formal language that is closed under negation and intersection, for which checking equality is decidable, and that supports tractable generation of instances.
2. PICK creates not one but a *family* of candidate formal expressions.
3. Driven by ideas from cognitive science (Section 6), instead of asking users to wrestle with these abstract formal expressions, PICK presents the user with a series of *concrete scenarios* (e.g., strings for regexes, traces for LTL, requests for access-control policies) that have been carefully chosen to *distinguish* among the candidates. The user is asked to indicate which scenarios they would accept and reject. As they classify concrete scenarios, they are actually implicitly classifying the underlying candidates. This process iterates until they have one or no candidate expressions left.

If in the end there is one candidate expression, then this is the best match with their intent. If there are no candidates left, that means either none of the expressions is a fit, or the user is not even consistent about their wishes. In either case, it would have been dangerous to take the single candidate produced by GenAI; instead, PICK forces them to clarify a poor GenAI input or, more fundamentally, an inconsistency in their own thinking.

We have applied PICK to three formal languages: regexes, access-control policies, and LTL. This paper addresses two research questions:

RQ1 To what extent does PICK's concrete-example–based workflow help users validate synthesized formal artifacts against their intended meaning, both alone and in contrast to direct inspection of candidate artifacts?

RQ2 What technical decisions support leveraging concrete examples and closure properties to identify correct formal artifacts from informal intent?

Sections 2 and 3 address RQ2 by presenting the design, algorithmic, and pragmatic choices underlying PICK. Section 4 addresses RQ1 through controlled user studies evaluating PICK across the three domains. In brief, our studies find that PICK significantly improves user accuracy across regexes and access-control policies, and enables novices to match domain-trained participants on LTL. Together, these results provide strong support that the PICK approach is well worthy of further consideration and use.

2 PICK: Design, Implementation, and Algorithmics

While we summarize the workflow here, Figure 2 explains the tool via a concrete example.

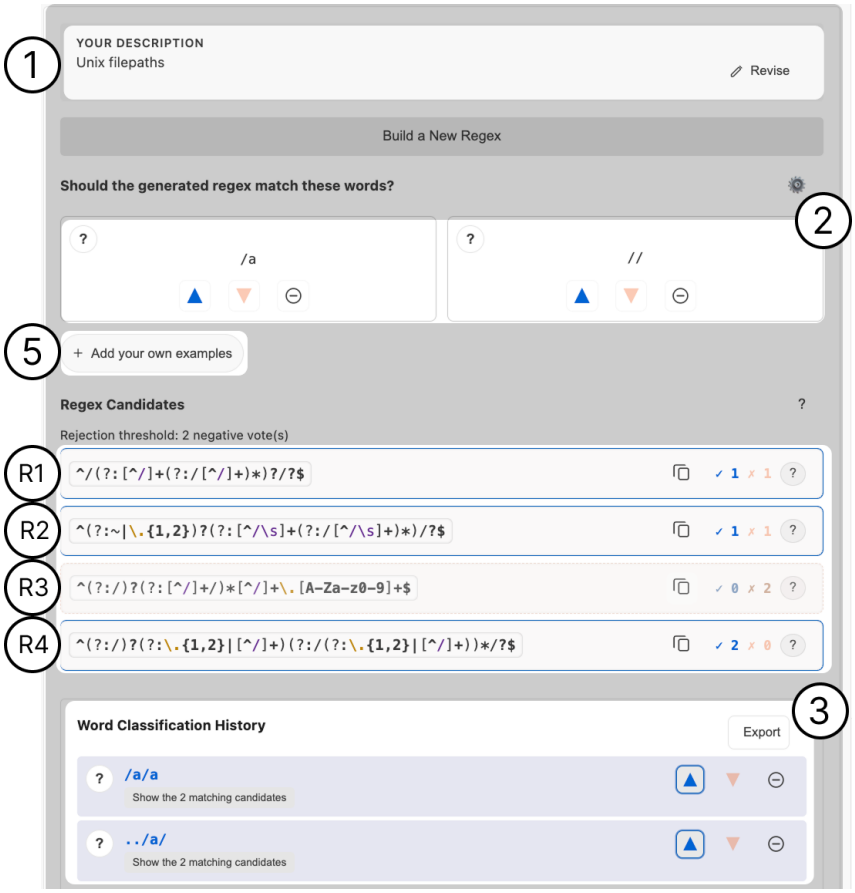


Figure 2 The PICK regex interface. The users' description of the desired regular expression ① is sent to a GenAI model, which yields four candidate regular expressions (R1) - (R4). PICK presents the user with distinguishing words ② and asks them to classify each one: upvoting classifies the word as matching the intended pattern, while downvoting classifies it as violating the pattern. Users can also mark that they are unsure about whether a scenario satisfies the pattern. The user has upvoted 2 scenarios ③, resulting in the elimination of (R3). Despite having received a Downvote, (R1) and (R2) remain in contention because they have not met the elimination threshold of 2 votes. Users can also add their own examples to help refine the candidates (⑤).

The user provides a textual description of the desired regex, which is sent to a GenAI model. As we discuss in Section 2.1, this yields a family of candidate expressions. PICK presents the user with pairs of scenarios (here, words) and asks them to classify each one: upvoting classifies the scenario as matching the intended pattern, while downvoting classifies it as violating the pattern. As scenarios are classified, candidates are either supported or eliminated, as we discuss in Section 2.2. This informs the generation of further scenarios.

Users can decide whether to see the candidate expressions from which they are *actually* selecting. In Figure 2, those outlined in blue remain in contention; those outlined in orange have been eliminated. Some of our studies showed the candidates while others did not; we discuss this in more detail in Section 4. PICK is driven by these candidates under the hood whether or not it shows them to the user.

PICK is functioning software. We have implemented it for all three languages—with essentially the same interface—to conduct user studies. We have deployed the regex version

as production software as a Visual Studio Code (VSC) extension that can be easily installed from the VSC Marketplace.[2] PICK leverages VSC APIs for accessing GenAI models.

2.1 Generating a Family of Candidates

We have discussed several reasons in Section 1 why it is problematic to have only one candidate expression. Thus, PICK generates a family of candidates to enable the rest of its workflow. There are many ways to obtain such a family:

1. By using different GenAI tools, which have different training and alignment procedures, we can obtain different interpretations of the original prose.
2. Even from a single GenAI, we can sample it for different candidate alternatives based on different readings (perhaps using higher temperature) of the original prose.
3. We can always use syntactic mutation operators to turn one expression into several.
4. We can sometimes do much better than syntactic mutation: in some cases we can create *semantic* mutants [62], which represent *plausible* alternatives corresponding to true, known human confusions (discussed in Section 4.4).

There are also practical considerations. For instance, the set of models available depends very much on what subscriptions a user has (and indeed, we would not be surprised if the various "free" tiers disappear as the cost of using models is no longer borne by funders). There may also be regional, political, and language considerations (e.g., for a person prompting in a language that is not English, a regional language model may be much better). PICK is therefore deliberately agnostic about how candidates are generated.

2.2 Supporting or Eliminating Candidates

In PICK, the user is superficially classifying scenarios, but they are actually voting on candidates, since the ultimate goal of the tool is to find the right candidate expression. Furthermore, while every scenario classification is obviously a statement about the candidate from which it is generated, it may also be a vote on the other candidates (which, being related, will share non-trivial overlaps in their languages). Thus every classification applies to all the candidates.

PICK maintains two scores for each candidate: Upvotes and Downvotes. When the user Accepts a scenario, this scenario is checked against every candidate; if it is a member of that candidate's language the candidate gets an Upvote, while if it is not a member of that candidate's language, the candidate gets a Downvote. Rejects are treated slightly differently. When the user Rejects a scenario, PICK again checks this against all the candidates. If the scenario is in the language of a candidate, that candidate is Downvoted. If the scenario is *not* in the language of a candidate, however, that candidate is *not* Upvoted. This is to prevent a candidate from being selected just because the user rejected all the other candidates; instead, we want the user to make affirmative decisions in favor of that candidate.

When a candidate gets two Downvotes, it is removed from contention. A candidate is eligible for selection only when it receives at least one Upvote (this number is configurable). Unsure votes do not impact elimination or selection.

[2] `https://marketplace.visualstudio.com/items?itemName=SiddharthaPrasad.pick-regex`

2.3 Computing Scenarios

The specific scenarios that users are asked to classify are computed from those candidates that are still in contention. When PICK is left with one candidate with sufficient support, or zero candidates, it terminates (Figure 1). So long as multiple candidates remain in contention, PICK uses formal language properties to generate two scenarios that highlight the features of or difference between the remaining candidate(s). It shows two scenarios rather than one due to considerations from cognitive science, which we discuss in Section 6.

Assume we have a universe of words U, where $\mathcal{L}(f) \subseteq U$ is the the language of an expression f (and $\neg\mathcal{L}(f) := U \setminus \mathcal{L}(f)$). PICK can be used with any formal language that supports these two properties:

1. Given a pair of candidate expressions A and B, we must be able to compute the set differences, $\mathcal{L}(A) \cap \neg\mathcal{L}(B)$ and $\mathcal{L}(B) \cap \neg\mathcal{L}(A)$. That is, the formal languages must be closed under negation and intersection.
2. Given one of the above sets, we must be able to sample for concrete instances of it: formally, membership should be decidable, but ideally there needs to be a generative process for producing such scenarios.

Beyond these formal requirements, there is also a *practical* one: the concrete scenarios drawn from these set differences should be easy for users to assess—that is, small, self-contained, and not requiring extensive cross-referencing to evaluate. This is not a property of the formal language alone, but of the relationship between the language and the domain it describes. We return to this point in Sections 7.1 and 7.2.

Broadly, the goal is to extract as much information as possible from each classification by choosing words whose acceptance patterns separate candidates (Section 3 describes some practical considerations).

This process begins by collapsing candidates with equivalent languages, exploiting the decidability of language equality and containment. With equivalent candidates removed, scenario generation focuses on elucidating genuinely distinct behaviors.

In the two-candidate case, with candidates A and B, the goal is to select scenarios whose acceptance differs between the two. If one language is a strict subset of the other—say $\mathcal{L}(A) \subset \mathcal{L}(B)$—then one scenario can be chosen that both candidates accept, while the other is accepted by B but rejected by A. If the languages partially overlap, then one scenario can be chosen from $\mathcal{L}(A) \setminus \mathcal{L}(B)$ and another from $\mathcal{L}(B) \setminus \mathcal{L}(A)$.

PICK generalizes this idea beyond 2 candidates by selecting scenarios whose acceptance patterns partition the candidate set. Each scenario in the pair assigns candidates to accepting or rejecting classes, so that each classification contributes information about how the user's judgments align with candidate behavior. We illustrate this concretely in Section 2.5. The full algorithm is shown in Algorithm 1.

2.4 Termination

The PICK process can have one of several outcomes:

1. The user converges on a single candidate with sufficient support.
2. The user eliminates all candidates.
3. The user's classifications are contradictory but keep falling within the range needed for acceptance or elimination.

The second outcome could arise in several ways: (a) perhaps the user made a simple mistake and accidentally misclassified a scenario; (b) perhaps the GenAI didn't generate any accurate candidates, due either to a poor problem statement or a mistake within the GenAI; or (c) perhaps the user wasn't actually clear on what they wanted and genuinely classified scenarios in ways that are inconsistent with the problem description.

The summary of the classifications at the bottom of the screenshot is designed to help with at least (a): users can easily review and reclassify scenarios, based on seeing them all presented together. For the other two cases, having the tool report that all candidates have been eliminated should alert the user that something has gone wrong. Whether the problem arises from the user or the GenAI, it would have been dangerous for the user to have blindly accepted raw GenAI output (as they might have done in the absence of PICK).

2.5 A Sample Run

We now walk through a sample run of PICK for "Unix filepaths", corresponding to Figure 2. The first column in the table below shows the classification round ("Setup" is the initial state; Figure 2 shows the tool's state at the start of round 2). The second column shows a concrete scenario (a string) and the third shows how the user reacted. The remaining columns show how the scores update: note that (a) one scenario can impact many candidates, and (b) accepting a candidate can impact both kinds of scores!

Round	Scenario	Decision	Upvotes				Downvotes			
			R1	R2	R3	R4	R1	R2	R3	R4
Setup			0	0	0	0	0	0	0	0
1	`/a/a`	Accept	1	0	0	1	0	1	1	0
	`../a/`	Accept	1	1	0	2	1	1	2	0
2	`/a`	Accept	2	1	0	3	1	2	3	0
	`//`	Reject	2	1	0	3	2	2	3	0

In Round 1, the user accepts two words. The first, "`/a/a`" is in both $\mathcal{L}$(`R1`) and $\mathcal{L}$(`R4`), but not in either $\mathcal{L}$(`R2`) or $\mathcal{L}$(`R3`). The second word, "`../a/`", is in $\mathcal{L}$(`R2`) and $\mathcal{L}$(`R4`), but not in either $\mathcal{L}$(`R1`) or $\mathcal{L}$(`R2`). Thus, at the end of Round 1, `R1` and `R2` have one Upvote and one Downvote each. `R3` has two Downvotes and `R4` has two Upvotes. Consequently, `R3` is eliminated from contention at the end of this round.

In Round 2 PICK generates two new scenarios to distinguish the remaining candidates: "`/a`" (in $\mathcal{L}$(`R1`) and $\mathcal{L}$(`R4`) but not in $\mathcal{L}$(`R2`)) and "`//`" (in $\mathcal{L}$(`R1`) but not in $\mathcal{L}$(`R2`) and $\mathcal{L}$(`R3`)). The user accepts "`/a`", which implicitly downvotes `R2`. This brings `R2`'s Downvote total to two, so it is eliminated. Next, the user rejects "`//`". This scenario is in $\mathcal{L}$(`R1`), so `R1` gets a Downvote, bringing its Downvote total to two. Thus, `R1` is also eliminated. As a result, PICK converges on `R4` as it has received at least one Upvote and is the only surviving candidate.

The reader might wonder why, in Round 2, we are updating the scores for `R3`, even though it has been eliminated. The reason is because of the ability to reclassify prior classifications (③ of Figure 2). Those reclassifications may revive a candidate, so we have to keep updating its score through all the classifications. Nevertheless, when PICK finishes, any candidate with two Downvotes is not part of the output set of candidates.

3 Design Choices in PICK

PICK, as presented in Section 2, embodies many design decisions, some of which are not evident from the screenshot. They speak to RQ2: the technical choices that arise when using

concrete examples and closure properties for intent validation. We discuss these below.

▶ *How many candidate expressions should* PICK *create?*

A good default seems to be about four. This creates enough variety to provide real choices, as Section 4 shows. A good number, however, really depends on how many *plausible* alternatives we can create. As we mention in Section 2.1, we can use semantic mutants to create several expressions based on known misconceptions that people have.

Every extra candidate beyond a point would seem to worsen the user experience; eventually users will get bored or overwhelmed. On the other hand, it is also important to consider every reasonable candidate. Since the reasonable candidates should be related, and votes impact all candidates, the total amount of work may be tractable. We return to this issue, with some evidence, in Section 7.2.

▶ *How much support (in terms of numbers of classified scenarios) do we want before eliminating an expression or confirming a final selection?*

As Section 2 mentions, we reject a candidate after two Downvotes, and require at least one Upvote for it to be selected. We chose these numbers based on two considerations:

1. We wanted to avoid one-off accidental selections from distorting the outcome too much. At the same time, we wanted the total number of steps to be tractable.
2. We ran several preliminary rounds of prototype user studies with regexes (Section 4.2.2) and carefully examined the decisions made by users relative to outcomes, and also read their justifications for their choices. While early rounds used a Downvote threshold of one, we observed that large percentages of participants were eliminating all of the candidates across both the *With List* and *Without List* conditions. When we changed the Downvote threshold to two, these percentages dropped noticeably (an average of 27% across all problems in both conditions; with a min of 4% and a max of 52%). We thus used two as the threshold for the remaining studies.

However, there is nothing especially canonical about these numbers beyond the basic principle of *robustness to imperfect user responses* over absolute minimalism. A user with a vested interest in the output may spend more time on each classification than our experimental participants, may better recognize subtle distinctions, or may desire higher assurance; accordingly, PICK allows these thresholds to be adjusted.

At the same time, the appropriate thresholds are also constrained by the semantics of the candidate space itself: when candidates overlap heavily, the amount of distinguishing evidence that can be obtained is inherently limited. For example, the regexes `a*` and `a+` differ only on the empty string. A fixed elimination threshold of two Downvotes would therefore yield a non-terminating decision process. As a result, when the number of distinguishing scenarios is less than the elimination threshold, PICK adjusts the threshold to the number of distinguishing scenarios between candidates.

▶ *Should we show the user the candidate formulae in addition to the scenarios?*

There are good arguments both for and against showing the candidate expressions in addition to the scenarios. For a user who is comfortable reading the candidates, working through the scenarios could feel tedious. Furthermore, an experienced user might calibrate the scenarios against the candidates, leading to more confidence in the final recommendation.

On the other hand, the candidates could distract a user from actually working with the examples. If the user does hold misconceptions about the language, seeing the candidates might reinforce those misconceptions and bias classification of the scenarios. In addition, while the candidates for languages like regexes and LTL tend to not be too large, for access-control policies they can be extremely large and hence simply would not fit on a screen.

The production version of PICK lets the user choose. From a research perspective, the question is whether showing or not showing the candidates affects the quality of the decisions. We therefore put this question to the test experimentally as part of our user studies (Sections 4.2.2 and 4.3.2), and found it made no difference.

▶ *What happens if a user runs out of candidates, or doesn't like the examples they're seeing?*
In such a situation, the user needs to revise the prompt. In principle, they can just start over with a revised version. However, doing so would throw away all the classification work they have already done. Therefore, PICK instead provides a notion of *revising* the statement (① in Figure 2). At a basic level, it provides the current prompt in editable form. Much more importantly, it retains all prior classifications and automatically re-applies them to each of the newly generated candidates. In practice, we have found this immensely valuable when refining prompts; without it, the tool can sometimes feel extremely frustrating. This is also a form of *robustness*: rather than forcing a selection from a bad candidate set, PICK tolerates the zero-candidate outcome (illustrated by the Dates question in Section 4.2.2).

▶ *What happens to semantically equivalent candidates?*
PICK clusters candidates by semantic equivalence. In principle, for the purpose of generating scenarios, it can throw away all but one per cluster. However, the different syntactic forms may be perceived differently by the user. They may prefer one over another because it's shorter, clearer, has better redundancy, etc. That is: syntax matters! Therefore, PICK keeps all the clustered formulae. At any point (especially when a cluster emerges as the winner), the user can look at all cluster members and decide which best meets their needs.

▶ *What about additional information provided by GenAI systems?*
Modern GenAI systems will often not only generate candidates but will also provide descriptive text about them: both a textual description of what the candidate is capturing, as well as a confidence score. While these are not necessarily reliable, they may be useful. Therefore, PICK retains this information, which can be viewed at any time.

In some cases, this auxiliary information takes the form of warnings. Because GenAI systems are trained across a wide range of programming tasks and formalisms, they can sometimes recognize when a task exceeds a given language's expressive power. For example, they may note that regular expressions cannot capture context-free patterns such as parsing HTML. When this occurs, PICK passes the warning on to the user.

▶ *Can we make richer use of scenarios?*
So far, scenarios are disconnected from the GenAI, which only produces candidates. When refining a prompt, however, PICK sends back the classifications the user has performed so far to the GenAI system. This additional guidance has the potential to improve the quality of subsequently generated candidates.

Of course, there is no guarantee the GenAI will accurately take these scenarios into account; however, as noted above, PICK immediately repeats the classification of new candidates, so any failures in this regard will be caught. PICK for regexes also provides a text box where users can write sample scenarios to accompany even the initial prompt.

PICK also uses other ways to obtain better scenarios. One is to ask the GenAI to also provide some interesting scenarios, which are added to the ones generated mechanically. The other is to allow users to edit the scenarios it is showing. Sometimes, seeing a machine-generated scenario inspires the user to amend it slightly to make it much more interesting. These are then also used during prompt revision, etc.

▶ *How do we balance information gain against responsiveness when generating scenarios?*
Because scenarios are drawn from the disagreement region between currently live candidates, every scenario is *informative* by construction: each classification necessarily rules on at

least one live candidate. In practice, however, there is a tension between finding ideal distinguishing examples and maintaining a productive interaction. Generating a theoretically optimal scenario can sometimes be computationally expensive, and waiting for such scenarios risks breaking the interactive flow of the tool.

This tradeoff is further justified by the structure of the candidate space itself. Heavy semantic overlap or containment among candidates can reduce the practical value of even theoretically distinguishing scenarios: a "good" example may simultaneously confirm many undesirable candidates. In such cases, perfect separation is less important than sustaining an interaction in which each classification contributes to clarifying the user's intent.

Accordingly, PICK favors timely progress over perfect *coverage* of the live candidate set. Informed by interaction design principles [53], scenario generation is capped at about one second in practice; beyond that point, the system prioritizes making forward progress over continuing to search for an optimal partition.

▶ *Why have an Unsure option?*

The Unsure option supports non-blocking exploration of the candidate space. Rather than requiring an immediate classification, PICK allows users to move past an example when they are uncertain, continue exploring other classifications, and return to it later if desired. Selecting Unsure marks the example as seen but does not affect candidate votes. As observed in our studies (Section 4), participants did make use of this option in practice.

▶ *How can PICK make differences between similar scenarios salient?*

PICK includes a diff mode (accessible via the gear icon beside ② in Figure 2) that highlights differences between scenario pairs. This is particularly useful when distinguishing scenarios that differ by only a small number of characters.

4 User Studies in Three Domains

We have described PICK and its many components. However, we have not demonstrated that the PICK idea is actually *useful*. We therefore conducted controlled studies to address RQ1: to what extent does PICK's workflow help users validate synthesized formal artifacts? The most natural question is whether PICK is actually effective. A particularly interesting special case is whether it helps in situations where the GenAI does not produce any correct answers. In addition, an important user interface element to examine is the impact of showing the candidate expressions, as discussed in Section 2.

We ran user studies with three formal languages: regexes, linear temporal logic [60] (LTL), and attribute-based access control [34, 29] (ABAC). We chose them because they not only enjoy the formal language properties we want, but also offer interesting contrasts:

- Regexes are widely used and can easily become quite complicated, even for seemingly simple patterns (as we showed in Section 1). The literature [45, 55, 59] also shows their potential to confuse.
- LTL formulae are used in settings as diverse as program analysis, verification, planning, and robotics. They have subtle semantics, as prior studies have established even with experienced LTL users [23, 26]. Using these results to generate candidate formulae provides a form of validity to our work. Furthermore, we can benchmark the performance of our users against those encountered in those studies.
- Access-control policies are interesting because their formal statements tend to be longer and more complex than for regexes or LTL. Whereas regexes and LTL entail reasoning about sequences or traces with limited information at each point, access control requires

Algorithm 1 Distinguishing candidate generation procedure. Each language is associated with an alphabet Σ and a universe of scenarios $U \subseteq \Sigma^*$. For any candidate f, $\Sigma_f \subseteq \Sigma$ denotes the set of alphabet symbols referenced by f.

Consumes a non-empty finite set of non-equivalent candidates R and a set of already-classified scenarios $E \subset U$. Produces 2 distinct scenarios.

1. Maintain list T of distinguishing scenarios, initially empty.
2. If $R = \{f_1, f_2, \ldots, f_n\}$ with $n \geq 2$, while $|T| < 2$, for all pairs $\{f_i, f_j\} \subseteq R$:
   a. Sample t from $((\mathcal{L}(f_i) \setminus \mathcal{L}(f_j)) \cup (\mathcal{L}(f_j) \setminus \mathcal{L}(f_i))) \setminus E$.
   b. If t is available, add it to T and E.
3. If $|R| = 1$ or $|T| < 2$, sample f from R.
   a. Sample distinguishing scenarios t^- from $\mathcal{L}(\neg f) \setminus E$ and t^+ from $\mathcal{L}(f) \setminus E$ if available.
   b. If t^+ is available, add it to T and E.
   c. If t^- is available, add it to T and E.
   d. If t^- is unavailable, try to sample scenario t_1^+ from $\mathcal{L}(f) \setminus (E)$. If t_1^+ is available, add it to T and E.
   e. If t^+ is unavailable, try to sample a scenario $t \in \Sigma_f^* \setminus E$. If t is available, add it to T and E.
   f. If $|T| < 2$, then no further distinguishing scenarios can be produced. Sample $2 - |T|$ scenarios from E if available, and add them to T.
4. Return the elements of T.

reasoning about multi-faceted relationships and attributes about entities, resources, and permissions. There is a risk that these aspects can challenge PICK.

We summarize what we learned in Section 4.5. Links to the online versions of the three studies are provided in Supplement Section 1.

4.1 Shared Study Logistics

Studies in all three domains had similar logistics and structure, which we describe before giving details of the individual studies.

Participants We ran all of our user studies on Prolific [63], offering participants a bit above minimum wage in the USA based on a time estimate for each task; payments were increased (including retroactively) when actual average times exceeded our estimates. These payments stabilized at USD 5 by the time we got to the LTL and ABAC tasks, which took about 30 minutes on average. For pilot rounds, we gathered data from 10 participants at a time. After each round we analyzed the data and responses for potential problems in the study design (ranging from instructions to problem setup to software) and revised until we were satisfied that the studies were functioning well. We then ran the formal study round, where we gathered data from 50 participants per condition, sometimes gathered over multiple sessions. To avoid familiarity becoming a factor, we prevented participants who had done a pilot round from doing the final round. We also prevented anyone who did the regex or LTL studies from doing the policy study. Across the pilot and study rounds, we spent a total of USD 3,266.66 on the studies.

Study Structure The study advertisements on Prolific followed a general format like "you'll be shown a description of <*kind of data*> and asked to classify <*kind of scenario*> based on

whether they match the description", with requirements on prior programming experience and access to a desktop or laptop. The full descriptions for all problems and conditions appear in Supplement Section 3. Our study did not fall under the aegis of our Institutional Review Board, but we applied standard precautions for safeguarding participants. A participant who accepted the task went first to a consent screen that explained what we collect (responses and demographics that Prolific requests), how we would use the data (academic research and publication), data storage (collection on an encrypted web server), anonymization (responses and demographics stored by anonymous ID), sharing (we could share a fully anonymized dataset with other researchers), and that participants could withdraw at any time.

Those who chose to consent and proceed next received screening questions to check for experience that had been advertised with the task. All participants were asked whether they had computer programming experience. Regex participants were also asked about their experience with regexes; access control and LTL participants were asked about their familiarity with Boolean logic. Participants who lacked any of these were screened out.

Accepted participants proceeded to a tutorial on both the task and the use of PICK. The tutorial explained the domain, gave an example of a scenario, and explained what it meant for a scenario to be (in)consistent with a prompt. For the LTL and ABAC scenarios, participants were then guided through 3–4 sample classifications with feedback to ensure they understood the setting. Regardless of score, participants could continue.

Finally, we took participants through a walkthrough of PICK, pointing out the various areas of the UI (as we did with readers in Section 2 when explaining Figure 2). At the end of the walkthrough, participants proceeded to see and respond to study problems. In all studies, we configured PICK to require a minimum of four upvotes for a candidate to be selected. Details of those problems are in the sections on each individual domain.

After a participant had completed all problems, we asked them to explain their reasoning on all scenarios for which their classification differed from that of our ground truth solution. These comments sometimes inspired modifications that we made while in the pilot phase for each problem. For the LTL and ABAC studies, we also asked participants whether they had prior experience with that formalism (radio button answer) and whether they had ever encountered or used tools that involve reasoning about temporal traces or access-control policies. We put these questions at the end to avoid invoking knowledge-based biases prior to the study.

Study Version of PICK All three studies were conducted with an earlier version of PICK. The final user-facing tool contains various pieces of functionality, such as scenario-generation timeouts and adaptive elimination thresholds, that are not germane to the study task. The visual interface is similar to that shown in Figure 2, with minor cosmetic differences: for example, Upvote and Downvote buttons were labeled "Accept" and "Reject" respectively. We therefore believe that the studies evaluate the core PICK workflow—the iterative classification of concrete scenarios to distinguish among candidates—and thus support the paper's main claims about the approach.

What We Did Not Study In practice, a user might give PICK a poorly worded description for which GenAI might not generate good candidates. We chose not to include such problems in our studies, since it was not clear what our analysis would need to look for in such cases. Poorly stated questions are in the mind of the beholder; poorly stated questions that we created might not have made sense to users, which would lack validity as a study design.

By design, our baselines compare PICK against unaided candidate selection rather than against other example-driven assistive tools; such tools adopt a different task decomposition, making a head-to-head comparison ill-defined (Section 6).

Accuracy When we report study results, we focus mainly on a participant's *accuracy* on each problem. A participant is accurate on an individual problem if they either converged on the correct answer or, if there was no correct candidate, they eliminated all candidates. The regex study featured one problem with no correct candidate.

Time is Not Reported While Prolific reports participant time-on-task, these data are not meaningful: we cannot tell whether participants interleaved doing our task with other activities. We looked at the reported time only to identify participants who appeared to not take the study seriously. We saw little evidence of this except in one case (Section 5).

4.2 Regular Expressions

Table 1 The problems in the Regular Expressions study. First we provide the **problem name** used in this paper. Then we show the problem description and the candidate expressions for that problem (as a ☐ list).

abWords: Words consist only of the letters `a` and `b`, have length at least two, and contain a `b` in every even position (2nd, 4th, 6th, ...).

- ☐ `(a|b){2,}`
- ☐ `((a|b)b)+`
- ☐ `((a|b)b)+(a|b)?` [**Correct**]
- ☐ `(a|b)(ab)*a?`

Times: Time in a 24-hour/military format (HH:MM). Note that both HH and MM must be two digits long, and padded with zeroes if needed. The time must also be valid.

- ☐ `([0-9]|1[0-9]|2[0-3]):[0-5][0-9]`
- ☐ `((0?[1-9])|(1[0-2])):([0-5]\d)(\s?((A|a|P|p)(M|m)))?`
- ☐ `([01][0-9]|2[0-3]):[0-5][0-9]` [**Correct**]
- ☐ `\d{2}:\d{2}`

Dates: Calendar dates in MM/DD/YYYY format. MM and DD must have two digits each and YYYY must have four digits. All should be padded with zeroes if needed. The date must also be valid. *The correct regex is not among the candidates.*

- ☐ `(0[1-9]|1[0-2])/(0[1-9]|[12][0-9]|3[01])/([0-9]{2})`
- ☐ `((01|03|04|05|06|07|08|09|10|11|12)/(0[1-9]|[12][0-9]|30)|(02)/(0[1-9]|1[0-9]|2[0-8]))/([0-9]{4})`
- ☐ `((01|03|05|07|08|10|12)/(0[1-9]|[12][0-9]|30)|(04|06|09|11)/(0[1-9]|[12][0-9]|3[01])|(02)/(0[1-9]|1[0-9]|2[0-8]))/([0-9]{4})`
- ☐ `\d{2}/\d{2}/\d{4}`

VarNames: Variable names that must start with a letter (uppercase or lowercase) or an underscore. After the first character, they may contain any number of letters, digits, or underscores.

- ☐ `([A-Za-z]|_+[0-9A-Za-z])\w*`
- ☐ `[A-Za-z][0-9A-Za-z]*`
- ☐ `\w+`
- ☐ `[A-Z_a-z]\w*` [**Correct**]

As we have discussed, regexes can be tricky to get right even for fairly simple patterns. The difference between $*$ and $+$ (zero occurrences and at least one occurrence, respectively) can be subtle, especially if people don't consider the zero case when imagining an expression. Some regexes get complicated when multiple sub-expressions are required to properly account for constraints within a pattern.

For our regex study, we wanted problems that would be more realistic than the typical problems in a theory of computation textbook, while also having enough subtlety that users might get them wrong. Table 1 shows the problems that we used. While the first one (abWords) is more of a classic textbook problem, we felt it would also provide a good baseline

Table 2 Accuracy on regex study. The first three columns report the percentage of participants within each condition who got each problem correct. The last three columns report the results of a chi-squared test between pairs of conditions (C for *Control*, L for *With List*, nL for *Without List*). Each of these cells lists χ^2, p. All comparisons had 1 degree of freedom and $N = 100$. Significant values ($p < \alpha = .05$) are in bold.

Problem	*Control*	*With List*	*Without List*	C-v-L	C-v-nL	L-v-nL
abWords	22.0%	60.0%	72.0%	**13.39, <.001**	**23.12, <.001**	1.11, .2912
Times	62.0%	66.0%	84.0%	0.04, .8350	**5.07, .0243**	3.41, .0647
Dates	26.0%	70.0%	66.0%	**17.67, <.001**	**14.53, <.001**	0.05, .8303
VarNames	50.0%	58.0%	58.0%	0.36, .5472	0.36, .5472	0.00, 1.0000

with classic regex operations. The other three were practical problems with different forms of subtlety: both the Times and Dates problems allow only certain specific values within their general format, while the VarNames problem has a restriction in the first position.

4.2.1 Generating Candidates and Scenarios

For each of these problems, we used our experience with regular expressions, as well as test cases, to determine the ground truth expression. We used a combination of methods to generate the alternative candidates. For the Times and Dates problems, we asked a GenAI to generate multiple candidates. For the abWords and VarNames problems, we manually created versions that captured common mistakes with these problems.

Rather than the information-gain-maximizing strategy used by PICK (Section 2.3), we employed an earlier, more naive, pairwise scenario generation procedure for this study. The procedure terminates after identifying a pair of scenarios that distinguishes any two active candidates, rather than maximizing across the full candidate set. Thus, it requires more human interaction than Algorithm 1.

4.2.2 Experiments and Results

Our regex study had participants in three different conditions:

1. The *Control* condition required participants to select a regex from a static list of candidates, without working through (classifying) examples. Participants could select an expression, indicate none was correct, or indicate that they were unsure of their answer. This condition corresponds to asking a GenAI to generate a set of candidates, then picking among them without additional tool support.
2. In the *With List* condition, participants could see the list of candidates as they classified scenarios (the version shown in Figure 2).
3. The *Without List* condition is identical to *With List*, except that PICK does not show the list of candidates. The contrast between *With List* and *Without List* helps us judge whether showing the list of candidates helps or hurts.

Table 2 reports on participant accuracy. Columns 2–4 report on accuracy as a percentage of participants (N=50 in each condition). Columns 5–7 report the χ^2 statistics and corresponding p-values from chi-squared tests between each pair of conditions. The accuracy percentages show that the control group did particularly poorly on two problems (abWords and Dates); the effect-size columns confirm that these differences were significant. For the Times and VarNames problems, the percentages identifying the expected expression were similar between

Table 3 Mean accuracy (%) by self-reported regex experience and perceived tool usefulness.

Experience	Very useless	Somewhat useless	Neutral	Not sure	Somewhat useful	Very useful
Never used	–	–	–	25% (n=1)	50% (n=3)	35% (n=5)
Occasional	25% (n=2)	25% (n=1)	42% (n=3)	50% (n=1)	40% (n=10)	40% (n=12)
Regular	–	50% (n=1)	–	–	50% (n=4)	39% (n=7)

the control and the other conditions (and indeed there is no significant difference). There were no significant differences between the *With List* and *Without List* conditions.

What might explain the low *Control* condition scores on abWords and Dates?

- For abWords, the most common incorrect answer (by 19 participants) was the third candidate, which only matches strings with even numbers of characters. Seven thought there was no correct regex, while another seven chose the fourth expression. Only one chose the first expression. Five were Unsure. The high rate of selection of the even-length regex (which at first glance seems correct) suggests that edge cases are easy to miss when evaluating candidates.
- The Dates problem, unlike the other three, had no correct candidate (though participants did not know this). As Table 1 shows, this problem had the longest candidate expressions. It is thus plausible that participants found these too onerous to work through manually, and thus simply made their best guess: among the ones who selected a candidate, 6 chose the first, 7 chose the second, 19 chose the third, and nobody chose the fourth. Five said they were Unsure (only two were unsure on both this and the abWords problem).

Of the 50 participants in the *Control* condition, 13 self-reported that they "regularly use or write regexes". **Not a single one of these 13 got both the abWords and the Dates problems correct**. This confirms the potential pitfalls of blindly selecting or approving candidates: examples can expose people to edge cases, especially when those examples have been selected to highlight the differences between candidates.

At the end of the *Control* condition study, participants were shown a screenshot of the PICK scenario-classification interface and asked how useful they felt such a tool would be. Table 3 shows the responses (columns) broken out by participants' self-reported experience using and writing regexes (rows). The majority of participants felt the tool would be useful, including all but one of the ones with the most experience. The two who felt the tool would be very useless each got only one problem correct.

These two pieces of data—improved accuracy on two problems for those who classified scenarios, and participants' opinions that examples would have been helpful—offer our first evidence that the PICK workflow offers benefits to users.

To List or Not to List As the last step of the study, participants in the *With List* and *Without List* conditions were asked about the (potential) value of seeing the candidates:

1. How useful "was it" (*With List*) / "would it be" (*Without List*) to see the candidate regex list? (5-point Likert from Very Useful to Very Useless, with "I'm not sure" option)
2. For each problem, would you have confidently chosen one from the candidate list **without** testing it on example words? (Options per problem were "yes", "no", and "unsure")

In each condition, 44 of 50 participants selected "somewhat" or "very useful". Among those with advanced regex experience, 8/10 (*With List*) and 9/14 (*Without List*) said they would

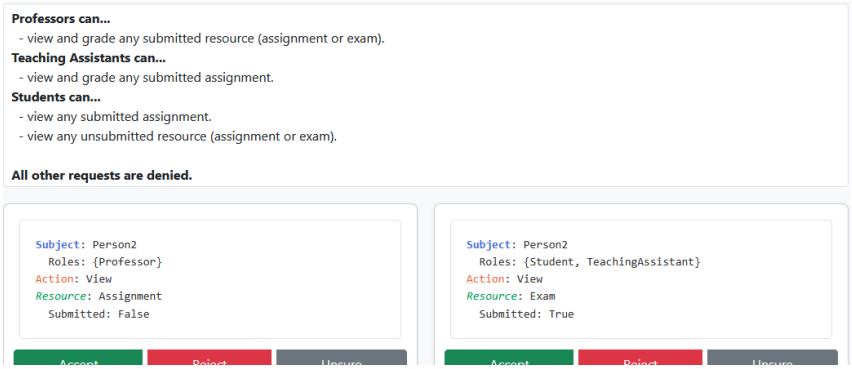


Figure 3 Our gradebook policy and two scenarios (requests), shown through the PICK UI. In the ABAC study, we applied a colorblind-friendly syntax highlighting scheme to the instructions, policies, and requests based on the recommendations of Patrignani [58].

be "very useful". Only 6/10 advanced participants in *With List* were confident in selecting an expression without reviewing scenarios on all problems; in *Without List*, only 2/14 advanced participants felt the same. These findings provide even more (albeit subjective) evidence from experts for PICK's design choice of grounding selection in concrete scenarios rather than candidate expressions alone.

4.3 Access-Control Policies

In attribute-based access control (ABAC), policies formalize the ability of *subjects* to take *actions* on *resources*, perhaps based on *attributes* of or relationships between these components. For example, consider a policy governing who can view and assign grades in a gradebook. The possible subjects are Faculty, Students, and TAs (for "teaching assistant"). The resource is an Assignment with an attribute indicating whether it has been Submitted, and the available actions are View and Grade.

When an entity wants to access a resource for a purpose, it issues a *request* comprised of a subject, action, resource, and attribute settings. Evaluating the policy against the data in the resource yields a *decision* (such as "permit" or "deny"). In the context of PICK, requests serve as scenarios. Figure 3 shows the gradebook policy and sample requests.

Reasoning about policies is especially subtle because of relationships that are easy to miss. One is *role overlap*, which occurs when one concrete subject holds multiple roles (a TA can also be a student), and readers could fail to account for those. Another can lie in having different policy rules render conflicting decisions based on different attributes. Industrial-strength policy languages (such as XACML [54]) include *policy combinators* to disambiguate such decisions.

We selected three problems based on the literature on analysis tools and our own personal experience. We intentionally made these problems modest in size and likely to have concise policies, because otherwise the *Control* condition would have been overwhelming. (In reality, policies can cover hundreds of roles [66], but our Prolific participants would likely not have been interested in reviewing policies that large.) Due to space constraints, we only summarize the three policy problems here. The full presentation of the policies and their candidates appear in Supplement Section 2.

- Accounting: This governs the ability of admins and accountants to read and edit financial and legal documents. Documents can be under audit or archived. Subjects can be in

Table 4 Accuracy on access control study. The first three columns report the percentage of participants within each condition who got each problem correct. The last three columns report the χ^2 statistics and corresponding p-values from chi-squared tests between each pair of conditions (C for *Control*, L for *With List*, nL for *Without List*. Significant values ($p < \alpha = .05$) are in bold.

Problem	*Control*	*With List*	*Without List*	C-v-L	C-v-nL	L-v-nL
Accounting	40%	66%	70%	**5.78, .0162**	**7.92, .0049**	0.05, .8303
Grading	46%	68%	66%	**4.08, .0434**	3.29, 0.0698	0.00, 1.0000
Technology	46%	60%	64%	1.45, .2293	2.59, .1078	0.04, .8368

training (in which case editing is not allowed).
- Grading: The policy shown in Figure 3.
- Technology: This governs the ability of network and system admins to access and edit firewalls and servers. Privileged actions can only be performed after hours if the subject is on call.

Each allows overlapping roles. Accounting features a resource with two attributes. Technology had attributes on each of the subject, action, and resource. These examples offered some structural variety, despite their relative simplicity as policies go.

4.3.1 Generating Candidates and Scenarios

For each problem, we wrote a prose version of the problem and asked a GenAI to produce a candidate policy to match that prose. Unsurprisingly, given the modest and textbook nature of these problems, these GenAI versions looked correct; after review, we made those our ground truth policies. We obtained the other candidates by modifying the ground truth policy systematically based on set relationships: one each that accepted a superset, a subset, and an overlapping but non-identical set of requests.

Scenario generation for policies follows Algorithm 1. The set of actions is usually fixed by the system. While the number of subjects and resources is finite, it is not fixed a priori; these policies gain their power from referring to classes of entities (such as "Faculty") and do not need to be updated as each one joins or leaves. However, most of these undistinguished entities are interchangeable. Therefore, it is very standard—and effective—to assume a reasonable size bound in formal methods tools for reasoning about such policies [27, 30, 31, 52]. By placing such a bound, the language of ABAC satisfies the properties needed by PICK, and a SAT-solver can decide the necessary questions.

4.3.2 Experiments and Results

As with the regex study, our ABAC studies had participants in three conditions—*Control*, *With List*, and *Without List*—with 50 participants in each. Table 4 summarizes the accuracy scores. Both conditions that classified scenarios significantly outperformed the *Control* group on the Accounting problem, as did the *With List* condition on the Gradebook problem. There were no significant differences between the *With List* and *Without List* conditions.

We noticed an interesting pattern in the incorrect answers: while only a few participants in the *Control* condition eliminated all the policies, participants in *each* of the other conditions (individually, not combined) were 2-4 times as likely to do so.[3] On the one hand, this suggests

[3] We did not compute the significance of these differences because we had not intended to do this

Table 5 The problems in the LTL study. Each problem was framed in terms of an instrument panel with three colored lights: Red, Green, and Blue. The variable `r` indicates when the red light is on, the variable `g` when the green light is on, and the variable `b` when the blue light is on. The ground truth formula for each problem is marked with a [**Correct**].

RedOnce: Red is on in exactly one state, not necessarily the first.

- ☐ `(!r) U (r & X(G(!r)))` [**Correct**]
- ☐ `F(X(r))`
- ☐ `G(F(r) -> X(G(!r))) & F(r)`
- ☐ `F(r & X(G(!r)))`
- ☐ `G (r -> XG!r)`
- ☐ `F(r) & (r -> X(G(!r)))`
- ☐ `F(r)`
- ☐ `X(F(r) U G(!r))`
- ☐ `(F(r) -> G(!r))`

RedOnOff: The Red light is on for zero or more states, and then turns off and remains off in the future.

- ☐ `(r U !r) & (G(!r -> G(!r)))` [**Correct**]
- ☐ `r U (!  r) & (!r -> X !r)`
- ☐ `r ->(X(r) | (G !(r)))`
- ☐ `F(G(!r))`
- ☐ `F(r -> X(G(!r)))`
- ☐ `G(r) U G(!r)`
- ☐ `F(r) & G(r -> X(U(!r)))`

RedBlue: Whenever the Red light is on, the Blue light will be on then or at some point in the future.

- ☐ `G(X(r) -> (X(b) | F(b)))`
- ☐ `G(r -> F(b))` [**Correct**]
- ☐ `r -> (F(b))`

that maybe our thresholds should be different. Much more interestingly, it means that PICK users are much more likely to end up with *no* solution, whereas *Control* users get a *wrong* solution that is similar to the correct one, but accepts a different set—and hence may be subtly wrong with the errors noticed only after a while.

Within the *Control* condition, participants who chose an incorrect policy appear to select the subset and superset candidates much more often than the overlapping policy. Further study would be required to determine whether some underlying cognitive factor explains this or whether it was just an artifact of our specific participants.

4.4 LTL

LTL is another language that has the properties we need. Its growing use in numerous domains beyond verification, such as robotics [2, 3, 6, 16, 28, 33, 36, 40, 69, 77, 78], not only means it has more users, but also that they may not be experts in its use. Such users are perhaps more likely to use GenAI and less able to judge the correctness of its output.

4.4.1 Generating Candidates and Scenarios

In principle, we can reproduce what we did for regexes and ABAC for LTL. For LTL, however, we have a unique opportunity to simulate PICK's workflow without even involving GenAI.

Researchers have extensively documented LTL misconceptions [23, 26]—for instance, assuming statements are globally quantified ("Implicit `G`"), or assuming that the antecedent

comparison when we started the study. In the interests of good research practice, we report it only as an interesting observation for further exploration, as we would have done had we formally pre-registered the study.

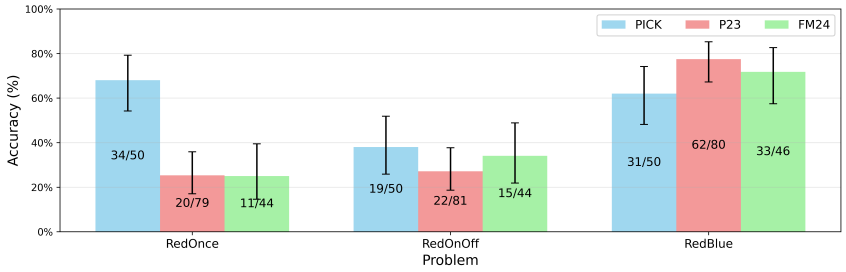


Figure 4 Accuracy comparison across problems with 95% CI (Wilson Score). Data for *P23* and *FM24* were sourced from publicly available datasets [25] and [24], respectively.

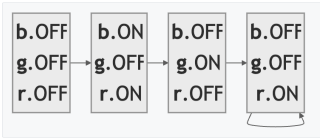
of an until (`U`) becomes false when the consequent becomes true. We refer to these two prior studies as *P23* [26] and *FM24* [23] throughout this section.

Their work provides two things of immediate value. First, it provides a set of example situations, which we adopted with only minor phrasing changes (to remove contextual dependencies). Second, in those studies, the participants—who knew LTL—provided actual LTL formulae (including a correct formula). We used these as our initial candidates instead of creating our own, since they capture realistic misconceptions.

Our study, in this situation, is therefore much richer than the previous two. We are armed with situations where even people with (significant) training and experience produced wrong formulae without realizing it. That means it is highly plausible that (a) a GenAI trained on such people might produce these formulae as output, and that (b) had such a formula been produced by GenAI, users might have accepted it. The question then is, how do our crowdsourced, non-experts armed with PICK fare compared to those experts?

The three problems—RedOnce, RedOnOff, and RedBlue—are listed in Table 5 along with all candidate formulae and the ground truth for each.

Scenarios for LTL are *traces*. A trace is a sequence of states, as shown on the right (here, each state specifies which of blue, green, and red lights are on or off at a moment in time). Arrows connect successive states, and every trace ends in a (perhaps multi-state) cycle (shown with a back-arrow), indicating that the final segment repeats forever. These traces were generated and sampled from candidates using the SPOT ω-automata manipulation and generation toolkit [13, 14].



4.4.2 Experiments and Results

We chose a set of formulas from *P23* and *FM24*, using *all* the candidates produced by the LTL-trained participants (the data are publicly available [24, 25]). These are shown in Table 5. Note that PICK participants were Prolific crowdworkers with no assumed LTL knowledge. All PICK participants used the tool in the *Without List* condition, since we were using the prior research data as our control group and we did not want to try to teach our potentially novice participants to read LTL.

Figure 4 reports the accuracy by problem. When testing for differences in accuracy between groups, Fisher's exact tests showed a clear effect on Problem RedOnce: novices using our tool outperformed participants in both prior studies on Problem RedOnce (PICK vs. FM24 and PICK vs. P23: Fisher's $p < 0.001$, OR ≈ 6.3). For the remaining problems,

neither comparison involving PICK was statistically significant (RedOnOff: $p = 0.83, 0.25$; RedBlue: $p = 0.39, 0.07$).

For more insight into whether novices reach performance levels close to trained participants, we applied a non-inferiority framework. Using the Miettinen-Nurminen score test [49] with a 10 percentage point margin, we tested whether novices were at most 10 points worse than trained participants (a pragmatic threshold, below which differences are unlikely to be practically meaningful given task variability). Results show that novices were non-inferior to *P23* on Problems RedOnce and RedOnOff and to *FM24* on Problem RedOnce, with inconclusive results elsewhere. Pooling across all three problems strengthens the conclusion: novices achieved 56.0% (84/150), compared to 43.3% (104/240) in *P23* and 44.0% (59/134) in *FM24*. Pooled non-inferiority tests confirm that novices using our tool are non-inferior to both prior studies (Fisher's $p < 0.001$ in both cases).

Overall, novices using our tool achieved accuracy within 10 percentage points of participants trained in LTL. This comparability holds problem by problem in two of the three cases and is strongly confirmed in the pooled analysis.

4.5 Summing Up

We believe that our user studies show clear value in PICK. In many cases we see statistically significant improvements. Even in cases where we do not, we almost always see higher performance percentages: that is, the lack of significance is not hiding poorer performance. This trend holds even when compared with *experts* on a domain (infinite-word temporal logics) known to be challenging. These results make a strong case for considering PICK as part of one's toolbox, even before considering the effect of large or complex candidates, which our study intentionally excluded.

There seems to be little to distinguish the *With List* and *Without List* conditions. There are many possible reasons for this, including the limitations of our participants. In a production tool, users would likely be frustrated to not be shown the candidates even on demand. At any rate, based on our evidence we cannot conclusively say whether showing the candidates helps or hurts performance.

5 Threats to Validity

We now discuss threats to two kinds of validity: both of the findings of this paper and, more importantly, the ability for PICK to be used in production.

Our study was conducted via crowdsourcing, which raises standard concerns about participant motivation, expertise, and engagement. We mitigated these risks by running test rounds to calibrate compensation (Section 4.1), using both Prolific and additional screening mechanisms, and collecting post-study familiarity surveys. These surveys indicate that many participants had relevant domain knowledge; several responses used technical terminology (e.g., references to Apache access files in the ABAC study). In both test and final rounds, we used completion times and written answers as signals of meaningful engagement.

For instance, we cannot rule out that participants may have used LLMs to assist their performance on the tasks. However, we ameliorated these concerns in two ways. The engagement signals just described (completion times and written responses) help flag participants who may have been outsourcing the task; during testing of the ABAC *Control* condition, for example, unusually short completion times suggested copying, and we addressed this by embedding policies as images. For the same reason, we presented LTL traces as images.

Even if participants *had* leaned on LLMs, however, the error rates reported in Tables 2 and 4 would reinforce rather than undermine the motivation for PICK.

Next, there is the threat from the size and variety of problems we gave. In the LTL case (Section 4.4), we were able to rely on a validated multi-year body of work. For regexes and ABAC, we chose what we hoped would be a usefully representative set of examples. In the ABAC case, however, these are only "textbook" examples; real policies are often significantly larger [57, 66]. However, we think this only *benefits* PICK: if a policy is hundreds of lines long, nobody is going to meaningfully look through several alternatives—even armed with a syntactic differencer—and make meaningful judgments. They are likely to want to resort to semantic differencing tools anyway (Section 6). However, even a generic semantic differencing tool might not zero in on instances designed for multi-way classification, so the user would have to manually perform what PICK does automatically.

That said, one significant threat in our *Control* and *With List* conditions is that the candidate expressions are presented as generic text. A user may well have sophisticated tools for navigating these. These tools may even help negotiate the large sizes of ABAC policies. Given such tools, the performances may improve. However, this is not inherently a problem: a user who has access to these tools and still wants to use PICK clearly finds benefit to our workflow, and if they can use it in conjunction with other tools to make better and faster decisions, we should rejoice!

Because we rely on decision procedures to generate scenarios, the time it takes to generate one could be problematic. In our studies, both regex and LTL generated new scenarios almost instantaneously, but ABAC uses a rough prototype that takes a few seconds. We did not expect crowdsourced participants to have the patience to wait that long, so we precomputed the scenarios. This would not be possible in a deployed system (and this precomputation inflates the pleasantness of the experience of the *With List* and *Without List* conditions). However, in this specific case, we were not engineering for speed, but rather used a system we were comfortable with; there may be other much more efficient systems. In general, however, we believe that a user who finds value in PICK would not mind waiting a few seconds for new scenarios. In particular since it takes several seconds *to form a judgment*, a production tool can just compute the next pairs *while the user is judging*, thereby eliminating the perception of this lag in most cases.

6 Foundational and Related Work

The value of concrete examples PICK draws on multiple theoretical foundations, some cognitive and some computational. Research in cognitive science shows that most people grasp abstract objects better when they approach it from concrete examples [4, 7, 51]. Formulae are inherently abstract: they represent a possibly infinite set of behaviors. People thus need access to concrete examples to help consider the abstract theories. This raises questions of where to get those examples, which examples would be most effective to show users, and how many examples to show at a time. These questions are at the heart of PICK.

The value of contrasting examples We build on the seminal theory of perceptual learning by Gibson and Gibson [21], which provided evidence that side-by-side contrast supports more precise discrimination among similar alternatives. In more recent years, the theories of *contrasting cases* [7, 64, 65, 67] and *variation theory* [42] speak to the selection and presentation of examples to help people understand abstract artifacts. They call for showing multiple examples side by side, with simultaneous examples carefully chosen to highlight differences in some feature of interest. They also establish the value of seeing both positive

and negative instances of an abstract concept. In both theories, working with carefully selected examples should help a learner develop a good mental model of the abstract object of study. Schwartz [68], a leading researcher in the field, summarizes as follows: "Contrasting cases are close examples that help people notice features they might otherwise overlook. They increase the precision and usability of knowledge". (Contrasting cases are widely used in daily life too: e.g., at food or wine tastings, where the pairings help people notice subtle differences.)

PICK asks users to classify scenarios in pairs whenever possible. Showing a pair of new scenarios should make it easier for users to focus on the differences. Since our scenarios are chosen specifically to highlight differences between the remaining candidates, this is an especially good fit. Our pairs are not fully aligned with contrasting case theory in that there is no guarantee that our algorithm chooses two scenarios that differ in a common underlying feature. Rather, we generate them in a principled way based on language containment relationships.

In addition, PICK always shows a summary of past classifications (③ of Figure 2). The hope is that seeing these classifications side-by-side helps people detect mistakes in their classification ("one of these is not like the other"). Therefore, we hope to exploit the perceptual system in this situation as well. (In our user studies, the number of reclassifications is small but not zero. Real users might reclassify much more.)

Specification-driven synthesis We see PICK as naturally situated within the classical, specification-driven synthesis tradition, dating back to work by Green, Waldinger, and Lee [22, 75, 76], in which a user provides a specification that defines the desired artifact. In PICK that specification is expressed in natural language and interpreted using a generative model, whose unreliability the system is explicitly designed to compensate for. Rather than treating natural language as a complete or correct specification, PICK uses it to generate an initial space of candidates and then refines that space through structured interaction.

Example-driven and interactive synthesis In many interactive synthesis systems, as systematized by Le, et al. [37], programs are synthesized from user-provided input-output examples. This approach has been taken for regexes (e.g., REGAE [81, 82], Graphite [56]) and for LTL (LtlTalk [20]). This form of interaction places the primary burden on users to *generate* concrete examples, often repeatedly, to drive the synthesis process. In contrast, PICK is designed around recognition-based interaction: users are primarily asked to judge concrete scenarios generated by the system, rather than to construct examples themselves. Recognition is generally easier than generation [74], and generating examples that effectively guide a synthesis algorithm can be especially challenging.

That said, PICK also supports example-driven input via an explicit interface for providing positive and negative examples (Figure 2). These examples are used to guide candidate generation and pruning, and are retained across prompt revisions.

Information-gain-driven question selection A complementary line of work studies information-driven interaction, aiming to improve synthesis by posing maximally informative questions to the user [32, 73]. PICK likewise derives information from distinguishing scenarios, but does not attempt to guarantee maximal information gain.

A key reason for this decision is that human intent is often elastic and judgments are imperfect. As our studies show, users sometimes misclassify scenarios, express uncertainty, or vote Unsure. In this setting, it is unclear how much practical benefit mathematically optimal question selection provides over strategies that yield merely good distinguishing examples. This observation motivates several robustness-oriented design choices in PICK, such as elimination thresholds (Section 3).

In addition, PICK does not rely solely on mechanically generated scenarios. As described in Section 3, it also incorporates scenarios suggested by the GenAI itself. While such scenarios may not always optimally distinguish between candidates, they can be more semantically meaningful. For example, a mechanically generated scenario such as `10-10-2012` could be an ideal candidate distinguisher in terms of regular-expression semantics, yet convey little information about where days and months are intended to appear. GenAI, however, may suggest a scenario such as `22-04-2012` that is less distinguishing under regular-expression semantics, but makes the intended placement of day and month fields explicit.

REGAE REGAE [81, 82] is the closest line of work to PICK, both because it targets regexes and because it is motivated by cognitive-science principles (in particular, the use of contrasting cases). It is firmly situated in the tradition of example-based interactive synthesis, and therefore inherits many of the associated tradeoffs, including the burden placed on users to construct informative examples.

Because REGAE does not integrate GenAI models into candidate generation, it must rely entirely on information conveyed through examples. Thus, the tool is ill-suited to tasks where the intended pattern depends on open-world semantic categories. For example, in the "countries of North America" task (Section 1), specifying the intended behavior by example requires complete enumeration, thereby eliminating the benefits of synthesis.

The key difference between REGAE and PICK is the level at which interaction takes place. REGAE repeatedly shuttles users between abstract candidate regexes and concrete examples: users must inspect regexes, supply examples to express preferences, and manually request distinguishing strings. While REGAE can generate examples (by either mutating previously seen strings or computing set differences between 2 selected regexes) these operations are invoked explicitly and not automatically integrated into an ongoing loop.

By contrast, PICK allows users to operate entirely at the level of concrete examples. The system automatically generates scenarios that distinguish candidates and incorporates user classifications directly into the interaction, without requiring users to inspect or compare abstract expressions. As a result, the workflow remains consistently grounded in concrete words, which our studies (Section 4) suggest is often more effective than requiring users to reason about abstract expressions.

Finally, REGAE is specialized to regexes and relies on regex-specific techniques such as DFA coverage for example generation, making its generality unclear. In contrast, we demonstrate PICK across three substantially different formal languages, suggesting that its core interaction paradigm applies beyond the regex domain.

Other Text-to-Language Tools In the paper we have already cited several tools that convert natural language to various formal languages, such as LTL. As noted, almost none of these tools use human feedback, often using (other) GenAI systems to check the output, which means poor specifications and systematic errors—such as misconceptions—will go uncaught. We know of two exceptions that do involve humans [10, 44], but both ask humans to judge *sub-formulae*, rather than concrete examples.

Change-Impact Systems A central feature of PICK is that it compares the *semantic* (as opposed to syntactic) differences between candidates. This is inspired by a line of work specifically in the policy analysis community [17, 30, 35, 38, 52]. Those works have argued for the value of using semantic differencing as a way of *exploring* policies, sometimes with an end to finding a preferable policy out of multiple candidate ones.

Those tools are not designed to cleanly handle a *set* of candidates: percolating decisions about scenarios to all relevant candidates, keeping scores on each candidate, running a decision-procedure loop, providing a way for users to revise decisions, and so on. A user

trying to employ those tools to our end would have to do all the steps in the PICK workflow entirely by hand. Furthermore, none of those tools (to our knowledge) has been subject to a formal user study of effectiveness of any sort. Arguably, our studies *justify* that line of work, showing that there was value to looking at concrete instances of differences. Finally, those works relate to our ABAC study, but to our knowledge those tools have not been applied to other languages like regexes and LTL. Thus, we can view PICK as a significant generalization of—but very much inspired by—that line of work.

Related Workflows Our workflow architecture (Figure 1) should be reminiscent of, and was definitely inspired by, other similar loops: Counterexample-Guided Abstraction Refinement (CEGAR) [9], which exploits language containment [5]; Counterexample-Guided Inductive Synthesis (CEGIS) [71]; and Reinforcement Learning from Human Feedback (RLHF) [8]. While we are loosely inspired by the CEGAR and CEGIS loops, our work is significantly different: we are not refining a property or program (though we are "refining" the set of candidates), and those loops do not involve humans, which ours crucially does. We are closer in spirit to RLHF, but with important differences. At a low level, the algorithmic details are vastly different. At a high level, in PICK the human in the loop is trying to generate a specific, single expression, not providing feedback that will be used to improve a separate general-purpose artifact.

7 Discussion

In this paper, we have tried to bring together three different ideas. We start with the utility but insufficiency of GenAI, and want to put humans back in the loop. To do so, we draw from the cognitive science literature on what they can most meaningfully do. To implement these ideas, we draw on formal language theory with a generic algorithmic framework that applies to a large family of in-use languages. Through a series of experiments across a broad range of domains—regexes, LTL, and ABAC—we show that there is real value in the PICK framework. We believe that the net result is a framework for the *responsible* use of GenAI, leveraging its strengths while using humans in targeted ways to compensate for its weaknesses.

7.1 Supporting SQL

Given the widespread use of SQL and the many GenAI-based text-to-SQL tools that have sprung up [50, 12, 19, 61], it is natural to wonder why we did not use PICK for SQL as well. In particular, for SQL, as for LTL, researchers have identified several misconceptions [46, 47, 48] that we could use to drive the generation of alternate formal statements from which users have to choose.

There are two reasons we have not considered SQL here. First, SQL does not precisely fit our theoretical needs: general query containment, for instance, is undecidable [1]. However, we can relax our precise requirements to obtain "good enough" results, which are still likely to be preferable to the alternative of a user rushing to adopt the first GenAI-produced query. The APEL system [84], for example, targets the natural-language-to-SQL setting: it generates candidate SQL queries from a natural language description and disambiguates them relative to a fixed database by producing distinguishing input–output examples, selected using Bayesian information gain, and asking users to judge the resulting query outputs. In this sense, APEL demonstrates that example-driven, human-in-the-loop disambiguation can be successfully applied to SQL despite its theoretical limitations.

The second, more fundamental challenge posed by SQL is that queries are meaningful only relative to a database. As a result, the effective object users must reason about is the pair of

a query and a dataset. SQL therefore does not readily satisfy the practical requirement noted in Section 2.3: it does not admit small, self-contained, contrasting "scenarios" of the kind used throughout this paper. Instead, query outputs are relational objects whose correctness often requires cross-referencing results against the underlying tables and reconstructing the intended semantics of the query, rather than making a localized judgment over a concrete example. This cognitive burden increases as queries or databases become more complex. In realistic deployment settings, query outputs may be very large (e.g., tables with many rows) or inherently opaque (e.g., numeric aggregates computed over large tables), making it unclear how users should distinguish between competing outputs from examples alone.

The issue, however, is not solely one of scale. Even when databases are small and carefully controlled, SQL's relational semantics can make outputs difficult to assess. The authors of APEL note that, despite using relatively small databases in their user studies, participants experienced difficulty assessing query outputs, especially those involving aggregations or other "onerous computations". Addressing this challenge robustly would require additional mechanisms, such as principled input reduction, output summarization, or visualization. These are sufficiently large additions that we consider them outside the scope of this paper but view them as exciting prospects for future work.

7.2 Concerns About Size

As noted in Section 2.3, PICK requires not only formal closure properties but also that scenarios be tractable for humans to judge. Two kinds of size concerns can arise: the number of scenarios, and the size of each one. The number of scenarios should be kept tractable quite naturally by the design of PICK. Our algorithm (Algorithm 1) is intentionally parsimonious in generating candidates: it only generates ones that are going to help discriminate between viable candidates and, after a small number of scenarios, the fate of each candidate is decided. Furthermore, even if we have many candidate formulae, they are likely to have some overlap, and again PICK counts votes against every eligible formula. We can confirm this experimentally: in the LTL studies, RedBlue had 3 candidate formulae, RedOnOff had 7, and RedOnce had 9. Participants who converged to the correct answer needed (median) only 8, 16, and 14 classifications, respectively (if we include everyone, even those who got the wrong answer, the medians drop to 8, 10, and 13). This lends credence to our hypothesis.

Crucially, the size of a scenario is not tied to the size of the candidate. ABAC is the clearest case: no matter how complex a policy grows, a scenario is still a request of fixed subject–action–resource shape. Regex and LTL have no such structural guarantee, but in practice the decoupling still tends to hold. The regexes and LTL formulae in Section 4 are derived from those used in practice, and yielded modestly sized scenarios for participants to judge. A regex for dates, however complicated, still distinguishes on short literals like `2026-03-31`, and large implicative LTL candidates (like those in Table 5) can be distinguished by traces of only 2–3 states. Therefore, in all these cases (and in other languages with similar characteristics), we do not envision the size of scenarios being a problem.

7.3 More about Regexes

The world of regexes is complicated: the same syntactic regex can have *different semantics* under different tools [41]! That is: one expression can have multiple semantics. This is the dual of what PICK does: multiple expressions under one semantics.

However, we believe our work can be extended for use in this case as well. There is no reason we cannot use multiple interpretation engines inside the system. For a given

expression, we can run it under the different engines. The challenge is that we can no longer directly exploit language closure properties, because we are—in effect—crossing languages. Instead, we can combine Mamouras, et al.'s formalization with sampling to obtain candidate scenarios that fall in the relevant sets.

There is a subtler issue in making the output actionable. In PICK, when we arrive at a single formula, for instance, we can simply present it to the user to deploy. In this alternate world, we could well converge on a formula—but our report would have to be along the lines of, "You have the right regular expression under the X engine, but not the Y engine". This is still useful to the user, who might otherwise have blindly used the "correct" regex but get incorrect results due to differences in the evaluation engines. Alternatively, the user would indicate what regex setting they are in, and if the modified PICK does not find a candidate in the desired engine, we would need additional GenAI runs to obtain a usable formula.

In short, we believe that with some modification, PICK can be used in domains where a fixed syntactic language has multiple (sometimes subtly different) semantics. In that setting, there is simply no difference to see in the syntax: it's entirely in the semantic layer. We therefore think a modified PICK can be especially useful in such settings.

7.4 Explore Just One?

PICK terminates when we converge on one candidate. But especially in mission-critical settings, a user might want to maximize understanding, not minimize work. Our algorithms can continue to function: they generate a scenario each from the candidate and its complement. Prior research [15] has shown the value of presenting "negative" scenarios to improve understanding. Our work does this in reverse: we use them to improve classification. Thus, PICK can continue to generate scenarios for the user to maximize their confidence.

7.5 Is PICK Strictly About GenAI?

The entire motivation of this paper has been to improve human interaction with GenAI output in a meaningful manner. But *all* forms of synthesis—whether classical, from formal specifications, or modern, using GenAI—have the same strength and weakness. Synthesis is a form of correct-by-construction, but that also implies *in*correct-by-construction. Verification addresses this issue by introducing redundancy (a separate property statement), which synthesis by definition eliminates. So how can we be sure that the user stated their intent correctly? The PICK process creates a form of redundancy, and hence consistency-checking, that all synthesis inherently lacks but ought to have. The same idea could, therefore, just as well be applied to traditional synthesis too!

This perspective also clarifies why PICK is not merely a response to current limitations of state-of-the-art generative models. As GenAI improves, it can be expected to produce higher-quality initial candidates. However, better candidates do not eliminate ambiguity in informal specifications: they simply shift user effort away from rejecting clearly incorrect outputs and toward examining alignment, edge cases, and subtle distinctions among plausible alternatives. For example, a prompt such as "Regex for dates" may yield increasingly accurate candidates as models improve, yet still leave unresolved questions about conventions such as date ordering, permitted formats, or calendar assumptions. In this sense, PICK is not chasing a moving target; improvements in GenAI directly strengthen, rather than weaken, the utility of the workflow.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Cambridge University Press, 1995. Corollary 6.3.2, Chapter 6.
2. Marco Antoniotti and Bud Mishra. Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers. In *ICRA*, pages 1441–1446. IEEE, 1995. doi:10.1109/ROBOT.1995.525480.
3. Brandon Araki, Xiao Li, Kiran Vodrahalli, Jonathan A. DeCastro, Micah J. Fry, and Daniela Rus. The logical options framework. In *ICML*, volume 139, pages 307–317. PMLR, 2021. URL: http://proceedings.mlr.press/v139/araki21a.html.
4. R. K. Atkinson, S. J. Derry, A. Renkl, and D. Wortham. Learning from examples: Instructional principles from the worked examples research. *Review of Educational Research*, 70(2):181–214, 2000. doi:10.3102/00346543070002181.
5. Felice Balarin and Alberto L. Sangiovanni-Vincentelli. An iterative approach to language containment. In Costas Courcoubetis, editor, *Computer Aided Verification*, pages 29–40, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.
6. Amit Bhatia, Lydia E. Kavraki, and Moshe Y. Vardi. Sampling-based motion planning with temporal goals. In *ICRA*, pages 2689–2696. IEEE, 2010. doi:10.1109/ROBOT.2010.5509503.
7. John D. Bransford, Jeffery J. Franks, Nancy J. Vye, and Robert D. Sherwood. *New Approaches to Instruction: Because Wisdom Can't Be Told.*, pages 470–497. Similarity and Analogical Reasoning. Cambridge University Press, New York, NY, US, 1989. doi:10.1017/CBO9780511529863.022.
8. Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4302–4310, Red Hook, NY, USA, 2017. Curran Associates Inc.
9. Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, September 2003. doi:10.1145/876638.876643.
10. Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification*, pages 383–396, Cham, 2023. Springer Nature Switzerland.
11. Christoph Czepa and Uwe Zdun. On the understandability of temporal properties formalized in Linear Temporal Logic, Property Specification Patterns and Event Processing Language. *IEEE Transactions on Software Engineering*, 46(1):100–112, 2020. doi:10.1109/TSE.2018.2859926.
12. Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Lu Chen, Jinshu Lin, and Dongfang Lou. C3: Zero-shot text-to-SQL with ChatGPT, 2023. URL: https://arxiv.org/abs/2307.07306, arXiv:2307.07306.
13. Alexandre Duret-Lutz. Manipulating LTL formulas using Spot 1.0. In *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*, pages 442–445. Springer, 2013. doi:10.1007/978-3-319-02444-8_31.
14. Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187. Springer, August 2022. doi:10.1007/978-3-031-13188-2_9.
15. Tristan Dyer, Tim Nelson, Kathi Fisler, and Shriram Krishnamurthi. Applying cognitive principles to model-finding output: The positive value of negative information. In *ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications*, 2022.

16 Georgios E. Fainekos, Hadas Kress-Gazit, and George J. Pappas. Temporal logic motion planning for mobile robots. In *ICRA*, pages 2020–2025. IEEE, 2005. doi:10.1109/ROBOT.2005.1570410.

17 K. Fisler, S. Krishnamurthi, L.A. Meyerovich, and M.C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 196–205, 2005. doi:10.1109/ICSE.2005.1553562.

18 Francesco Fuggitti and Tathagata Chakraborti. nl2ltl–a Python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 16428–16430, 2023.

19 Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145, January 2024. doi:10.14778/3641204.3641221.

20 Ivan Gavran, Eva Darulova, and Rupak Majumdar. Interactive synthesis of temporal specifications from examples and natural language. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–26, 2020.

21 James J. Gibson and Eleanor J. Gibson. Perceptual learning: Differentiation or enrichment? *Psychological Review*, 62(1):32–41, 1955.

22 Cordell C. Green. Application of theorem proving to problem solving. In *International Joint Conference on Artificial Intelligence*, 1969.

23 Ben Greenman, Siddhartha Prasad, Antonio Di Stasio, Shufang Zhu, Giuseppe De Giacomo, Shriram Krishnamurthi, Marco Montali, Tim Nelson, and Milda Zizyte. Misconceptions in finite-trace and infinite-trace linear temporal logic. In *International Symposium on Formal Methods*, pages 579–599. Springer, 2024.

24 Ben Greenman, Siddhartha Prasad, Antonio Di Stasio, Shufang Zhu, Giuseppe De Giacomo, Shriram Krishnamurthi, Marco Montali, Tim Nelson, and Milda Zizyte. Artifact for misconceptions in finite-trace and infinite-trace linear temporal logic, July 2024. doi:10.5281/zenodo.12770102.

25 Ben Greenman, Sam Saarinen, Tim Nelson, and Shriram Krishnamurthi. Accepted Artifact for Little Tricky Logic: Misconceptions in the Understanding of LTL, August 2022. doi:10.5281/zenodo.6988909.

26 Ben Greenman, Sam Saarinen, Tim Nelson, and Shriram Krishnamurthi. Little tricky logic: Misconceptions in the understanding of LTL. *Programming*, 7(2):7:1–7:37, 2023. doi:10.22152/programming-journal.org/2023/7/7.

27 Dimitar P. Guelev, Mark Ryan, and Pierre Yves Schobbens. Model-checking access control policies. In Kan Zhang and Yuliang Zheng, editors, *Information Security*, pages 219–230, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

28 David Gundana and Hadas Kress-Gazit. Event-based signal temporal logic synthesis for single and multi-robot tasks. *IEEE Robotics and Automation Letters*, 6(2):3687–3694, 2021. doi:10.1109/LRA.2021.3064220.

29 Vincent C. Hu, D. Richard Kuhn, David F. Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015. doi:10.1109/MC.2015.33.

30 Graham Hughes and Tevfik Bultan. Automated verification of access control policies using a SAT solver. *Int. J. Softw. Tools Technol. Transf.*, 10(6):503–520, December 2008.

31 Daniel Jackson. Automating first-order relational logic. In *ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, 2000.

32 Ruyi Ji, Jingjing Liang, Yingfei Xiong, Lu Zhang, and Zhenjiang Hu. Question selection for interactive program synthesis. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 1143–1158, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3385412.3386025.

33 Yiannis Kantaros and Michael M. Zavlanos. STyLuS*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. *International Journal of Robotics Research*, 39(7):812–836, 2020. doi:10.1177/0278364920913922.

34 D. Richard Kuhn, Edward J. Coyne, and Timothy R. Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010. doi:10.1109/MC.2010.155.

35 Selasi Kwashie, Wei Kang, Sandeep Santhosh Kumar, Geoff Jarrad, Seyit Camtepe, and Surya Nepal. Acumen: Analysing the impact of organisational change on users' access entitlements. In *Computer Security – ESORICS 2023: 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25–29, 2023, Proceedings, Part IV*, page 410–430, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-51482-1_21.

36 Morteza Lahijanian, Shaull Almagor, Dror Fried, Lydia Kavraki, and Moshe Vardi. This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In *AAAI*, pages 3664–3671. AAAI Press, 2015. URL: https://shaull.github.io/pub/LAFKV15.pdf.

37 Vu Le, Daniel Perelman, Oleksandr Polozov, Mohammad Raza, Abhishek Udupa, and Sumit Gulwani. Interactive program synthesis, 2017. URL: https://arxiv.org/abs/1703.03539, arXiv:1703.03539.

38 Dan Lin, Prathima Rao, Elisa Bertino, and Jorge Lobo. An approach to evaluate policy similarity. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, SACMAT '07, page 1–10, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1266840.1266842.

39 Jason Xinyu Liu, Ziyi Yang, Benjamin Schornstein, Sam Liang, Ifrah Idrees, Stefanie Tellex, and Ankit Shah. Lang2LTL: Translating natural language commands to temporal specification with large language models. In *Workshop on Language and Robotics at CoRL 2022*, 2022. URL: https://openreview.net/forum?id=VxfjGZzrdn.

40 Savvas G. Loizou and Kostas J. Kyriakopoulos. Automatic synthesis of multi-agent motion tasks based on LTL specifications. In *CDC*, pages 153–158. IEEE, 2004. doi:10.1109/CDC.2004.1428622.

41 Konstantinos Mamouras, Alexis Le Glaunec, Wu Angela Li, and Agnishom Chattopadhyay. Static analysis for checking the disambiguation robustness of regular expressions. *Proc. ACM Program. Lang.*, 8(PLDI), June 2024. doi:10.1145/3656461.

42 Ference Marton. *Necessary Conditions of Learning*. Routledge, 2014.

43 Allison McCoy, Eric Thomas, Marie Krousel-Wood, and Dean Sittig. Clinical decision support alert appropriateness: A review and proposal for improvement. *The Ochsner Journal*, 14:195–202, June 2014.

44 Daniel Mendoza, Christopher Hahn, and Caroline Trippel. Translating natural language to temporal logics with large language models and model checkers. In *2024 Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–11, 2024. doi:10.34727/2024/isbn.978-3-85448-065-5_17.

45 Louis G. Michael, James Donohue, James C. Davis, Dongyoon Lee, and Francisco Servant. Regexes are hard: Decision-making, difficulties, and risks in programming regular expressions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 415–426, 2019. doi:10.1109/ASE.2019.00047.

46 Daphne Miedema, Efthimia Aivaloglou, and George Fletcher. Identifying SQL misconceptions of novices: Findings from a think-aloud study. In *Proceedings of the 17th ACM Conference on International Computing Education Research*, ICER 2021, page 355–367, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3446871.3469759.

47 Daphne Miedema, Michael Liut, George Fletcher, and Efthimia Aivaloglou. MSMI1: Towards a validated SQL misconceptions instrument. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 2*, ICER '23, page 16–17, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3568812.3603471.

48 Daphne Miedema, Michael Liut, George H. L. Fletcher, and Efthimia Aivaloglou. "There is no ambiguity on what to return": Investigating the prevalence of SQL misconceptions. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, Koli Calling '23, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3631802.3631821.

49 Olli Miettinen and Markku Nurminen. Comparative analysis of two rates. *Statistics in Medicine*, 4(2):213–226, 1985.

50 Ali Mohammadjafari, Anthony S. Maida, and Raju Gottumukkala. From natural language to SQL: Review of LLM-based text-to-SQL systems, 2025. URL: https://arxiv.org/abs/2410.01066, arXiv:2410.01066.

51 Arseny Moskvichev, Roman Tikhonov, and Mark Steyvers. Teaching categories via examples and explanations. *Cognition*, 238:105511, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0010027723001452, doi:10.1016/j.cognition.2023.105511.

52 Timothy Nelson, Christopher Barratt, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. The Margrave tool for firewall analysis. In *Proceedings of the 24th International Conference on Large Installation System Administration*, LISA'10, page 1–8, USA, 2010. USENIX Association.

53 Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

54 OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0. https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2013. OASIS Standard.

55 Olaperi Yeside Okuboyejo, Sigrid Ewert, and Ian Sanders. Goofs in the class: Students' errors and misconceptions when learning regular expressions. In George Wells, Monelo Nxozi, and Bobby Tait, editors, *ICT Education*, pages 57–71, Cham, 2021. Springer International Publishing.

56 Cyrus Omar, Young Seok Yoon, Thomas D LaToza, and Brad A Myers. Active code completion. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 859–869. IEEE, 2012.

57 Simon Parkinson and Saad Khan. A survey on empirical security analysis of access-control systems: A real-world perspective. *ACM Comput. Surv.*, 55(6), December 2022. doi:10.1145/3533703.

58 Marco Patrignani. Why should anyone use colours? or, syntax highlighting beyond code snippets, 2021. URL: https://arxiv.org/abs/2001.11334, arXiv:2001.11334.

59 Nelishia Pillay. Learning difficulties experienced by students in a course on formal languages and automata theory. *SIGCSE Bull.*, 41(4):48–52, January 2010. doi:10.1145/1709424.1709444.

60 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32.

61 Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

62 Siddhartha Prasad, Ben Greenman, Tim Nelson, and Shriram Krishnamurthi. A misconception-driven adaptive tutor for linear temporal logic. In Ruzica Piskac and Zvonimir Rakamarić, editors, *Computer Aided Verification*, pages 185–200, Cham, 2025. Springer Nature Switzerland.

63 Prolific. Prolific. https://www.prolific.com, 2025. London, UK. Accessed April 2025.

64 B. Rittle-Johnson and J. Star. Does comparing solution methods facilitate conceptual and procedural knowledge: An experimental study on learning to solve equations. *Journal of Educational Psychology*, 99:561–574, 2007. doi:10.1037/0022-0663.99.3.561.

65 B. Rittle-Johnson and J. R. Star. Compared with what? The effects of different comparisons on conceptual knowledge and procedural flexibility for equation solving. *Journal of Educational Psychology*, 101(3):529–544, 2009. doi:10.1037/a0014224.

66 Andreas Schaad, Jonathan Moffett, and Jeremy Jacob. The role-based access control system of a European bank: A case study and discussion. In *Proceedings of the Sixth ACM Symposium*

on Access Control Models and Technologies, SACMAT '01, page 3–9, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/373256.373257.

67 Daniel L. Schwartz, Catherine C. Chase, Marily A. Oppezzo, and Doris B. Chin. Practicing versus inventing with contrasting cases: The effects of telling first on learning and transfer. *Journal of Educational Psychology*, 103(4):759–775, 2011.

68 Daniel L. Schwartz, Jessica M. Tsang, and Kristen P. Blair. *The ABCs of How We Learn: 26 Scientifically Proven Approaches, How They Work, and When to Use Them*. W.W. Norton & Company, Inc, 2016.

69 Ankit Shah, Pritish Kamath, Julie A. Shah, and Shen Li. Bayesian inference of temporal task specifications from demonstrations. In *NeurIPS*, pages 3808–3817, 2018.

70 Heleen Sijs, Jos Aarts, Arnold Vulto, and Marc Berg. Overriding of drug safety alerts in computerized physician order entry. *Journal of the American Medical Informatics Association : JAMIA*, 13:138–147, March 2006. doi:10.1197/jamia.M1809.

71 Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodík, and Kemal Ebcioğlu. Programming by sketching for bit-streaming programs. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, page 281–294, New York, NY, USA, 2005. Association for Computing Machinery. doi:10.1145/1065010.1065045.

72 Shahroz Tariq, Mohan Baruwal Chhetri, Surya Nepal, and Cecile Paris. Alert fatigue in security operations centres: Research challenges and opportunities. *ACM Comput. Surv.*, 57(9), April 2025. doi:10.1145/3723158.

73 Ashish Tiwari, Arjun Radhakrishna, Sumit Gulwani, and Daniel Perelman. Information-theoretic user interaction: Significant inputs for program synthesis, 2020. URL: https://arxiv.org/abs/2006.12638, arXiv:2006.12638.

74 Endel Tulving. *Elements of Episodic Memory*. Oxford University Press, Oxford, 1983.

75 Richard J. Waldinger. *Constructing Programs Automatically Using Theorem Proving*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1969.

76 Richard J. Waldinger and Richard C. T. Lee. PROW: A step toward automatic program writing. In *Proceedings of the First International Joint Conference on Artificial Intelligence*, pages 241–252. Morgan Kaufmann, 1969.

77 Yanwei Wang, Nadia Figueroa, Shen Li, Ankit Shah, and Julie Shah. Temporal logic imitation: Learning plan-satisficing motion policies from demonstrations. In *Conference on Robot Learning, CoRL*, pages 94–105. PMLR, 2022. URL: https://proceedings.mlr.press/v205/wang23a.html.

78 Tichakorn Wongpiromsarn, Alphan Ulusoy, Calin Belta, Emilio Frazzoli, and Daniela Rus. Incremental temporal logic synthesis of control policies for robots interacting with dynamic agents. In *IROS*, pages 229–236. IEEE, 2012. doi:10.1109/IROS.2012.6385575.

79 Yilongfei Xu, Jincao Feng, and Weikai Miao. Learning from failures: Translation of natural language requirements into linear temporal logic with large language models. In *2024 IEEE 24th International Conference on Software Quality, Reliability and Security (QRS)*, pages 204–215, 2024. doi:10.1109/QRS62785.2024.00029.

80 Mian Yang, Vijayalakshmi Atluri, Shamik Sural, and Ashish Kundu. Extraction of machine enforceable ABAC policies from natural language text using LLM knowledge distillation. In *Proceedings of the 30th ACM Symposium on Access Control Models and Technologies*, SACMAT '25, page 157–168, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3734436.3734447.

81 Tianyi Zhang, Zhiyang Chen, Yuanli Zhu, Priyan Vaithilingam, Xinyu Wang, and Elena L. Glassman. Interpretable program synthesis. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3411764.3445646.

82 Tianyi Zhang, London Lowmanstone, Xinyu Wang, and Elena L. Glassman. Interactive program synthesis by augmented examples. In *Proceedings of the 33rd Annual ACM Symposium on*

User Interface Software and Technology, UIST '20, page 627–648, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3379337.3415900.

83 Mengyan Zhao, Ran Tao, Yanhong Huang, Jianqi Shi, Shengchao Qin, and Yang Yang. NL2CTL: Automatic generation of formal requirements specifications via large language models. In Kazuhiro Ogata, Dominique Mery, Meng Sun, and Shaoying Liu, editors, *Formal Methods and Software Engineering*, pages 1–17, Singapore, 2024. Springer Nature Singapore.

84 Ruiqi Zhong, Charlie Snell, Dan Klein, and Jason Eisner. Non-programmers can label programs indirectly via active examples: A case study with text-to-SQL. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5126–5152, Singapore, December 2023. Association for Computational Linguistics. URL: https://aclanthology.org/2023.emnlp-main.312/, doi:10.18653/v1/2023.emnlp-main.312.