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

Siddhartha Prasad
Brown University, Providence, RI, USA

Skyler Austen
Brown University, Providence, RI, USA

Kathi Fisler
Brown University, Providence, RI, USA

Shriram Krishnamurthi
Brown University, Providence, RI, USA

Abstract

This artifact accompanies the paper *Meaningful Human-in-the-Loop Checking of GenAI Synthesis for Restricted Languages*. It provides (1) the PICK VS Code extension for regex synthesis with human-in-the-loop validation, and (2) anonymized user-study data and Docker images that reproduce the study interfaces and statistical analyses reported in the paper.

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/DARTS.12.1.7

Related Article Siddhartha Prasad, Skyler Austen, Kathi Fisler, and Shriram Krishnamurthi, "Meaningful Human-in-the-Loop Checking of GenAI Synthesis for Restricted Languages", in 40th European Conference on Object-Oriented Programming (ECOOP 2026), LIPIcs, Vol. 372, pp. 22:1–22:31, 2026. https://doi.org/10.4230/LIPIcs.ECOOP.2026.22

Related Conference 40th European Conference on Object-Oriented Programming (ECOOP 2026), June 29–July 3, 2026, Brussels, Belgium

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2026 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

The paper makes the following contributions; each is linked to the artifact component that supports it.

1. PICK is functioning software, deployed as a VS Code extension for regexes (paper section 2). appendix B provides the extension itself, installable via a bundled VSIX or GitHub Codespaces, so reviewers can confirm the iterative candidate-generation and scenario-classification workflow described in the paper.
2. Evidence that PICK's concrete-example-based workflow helps users validate synthesized formal artifacts against their intended meaning (paper section 4). appendix C contains anonymized data from our user studies and reproduces the results reported in the paper.
   - appendix D describes how to reproduce the regex study interface (paper section 4.2).
   - appendix E describes how to reproduce the access-control policy study interface (paper section 4.3).
   - appendix F describes how to reproduce the LTL study interface (paper section 4.4).

Dagstuhl Artifacts Series
Schloss Dagstuhl – Leibniz-Zentrum für Informatik,
Dagstuhl Publishing, Germany

2 Content

The artifact archive contains:

- `pick-regex.vsix` – the PICK VS Code extension (appendix B); code artifact.
- `regex-study-ecoop-artifact.tar` – Docker image for the regex study (appendix D); code artifact.
- `policy-study-ecoop-artifact.tar` – Docker image for the ABAC study (appendix E); code artifact.
- `policy-control-ecoop-artifact.tar` – Docker image for the ABAC control condition (appendix E); code artifact.
- `ltl-study-ecoop-artifact.tar` – Docker image for the LTL study (appendix F); code artifact.
- `data-analysis-ecoop-artifact.tar` – Docker image for study data analysis (appendix C); code artifact.
- `user-study-raw-data/` – anonymized raw CSV data from all user studies (appendix C); data artifact, CSV format.
- `qualtrics-regex-control.pdf` – the Qualtrics survey used for the regex *Control* condition (appendix D); data artifact, PDF format.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS).

In addition, the artifact is also available at: `https://github.com/sidprasad/pick-regex`.

4 Tested platforms

Hardware. No special hardware is required. Any modern laptop or desktop with at least 8 GB RAM and 20 GB free disk space is sufficient.

Software.

- Either Visual Studio Code $\geq$ 1.95 with a language-model provider (e.g. GitHub Copilot), or a GitHub account with access to GitHub Codespaces (appendix B).
- Docker (appendix C).
- A web browser (appendix C).

Estimated time. Setup: approximately 5 minutes. Kick the tires: approximately 15 minutes.

5 License

The artifact is available under the MIT License.

6 MD5 sum of the artifact

2f78bb3882dca87bb316662095cfe3c6

7 Size of the artifact

958 MB

A Getting Started

Unpack the artifact archive:

```
mkdir artifact && tar xzf ecoop-2026-artifact.tar.gz -C artifact
cd artifact
```

A.1 Kick the Tires (15 minutes)

1. **Install and run the pick regex tool.** Follow the instructions in appendix B (Option A or Option B) to install the VS Code extension. Enter a natural-language prompt (e.g. *"unix filepaths"*) and confirm that candidates are generated and that the upvote/downvote workflow functions as described.
2. **Reproduce tables and statistics from user study data.** Follow the instructions in appendix C to review the analysis conducted to produce the tables and figures in the paper.

B PICK Regex Tool (VS Code Extension)

This section describes how to access the PICK regex tool shown in paper fig. 2. PICK is a *VS Code extension*, not an LLM or GenAI application per se: it is a developer tool that happens to use a language model in the back end (via the VS Code Language Model API, `vscode.lm`) to generate candidate regexes. The extension is therefore not tied to any particular model or provider – it works with whichever LM provider is available in VS Code (e.g. GitHub Copilot).

We provide two ways to access the tool.

Option A – Local VSIX install. If you are a regular VS Code user and have a language-model provider enabled (e.g. GitHub Copilot), you can install the extension directly:

1. Install the bundled `.vsix` file included in this artifact (see Installing from a VSIX for detailed instructions):

   ```
   code --install-extension pick-regex.vsix
   ```

   Alternatively, the extension is published on the VS Code Marketplace at `https://marketplace.visualstudio.com/items?itemName=SiddharthaPrasad.pick-regex` and can be installed directly from within VS Code by searching for `pick-regex` in the Extensions view.
2. If you do not already have a language-model provider, install the GitHub Copilot extension and sign in with a GitHub account. **GitHub Copilot Free** is included at no cost with any GitHub account and provides access to models (e.g. GPT-4o) with rate limits more than adequate for PICK.

Option B – GitHub Codespaces (zero local setup). If you are not a regular VS Code user, we also provide a running instance via GitHub Codespaces. The repository includes a `.devcontainer` configuration that pre-installs both the PICK extension and GitHub Copilot, so no local toolchain is required. Navigate to `https://codespaces.new/sidprasad/pick-regex` to create a Codespace with everything pre-configured. This option requires a **GitHub account**. GitHub's free tier provides 120 core-hours/month and 15 GB storage, which is more than sufficient for artifact evaluation; however, usage beyond the free tier may incur cost.

Using the tool. Once the extension is installed (via either option), open the PICK panel in VS Code and ensure a model is selected in the dropdown. Type a natural-language prompt describing the regex you want – for example, you could try *"unix filepaths"*. Because candidates are generated by a language model, you may not obtain the exact same candidates shown in the paper; however, you can confirm that the tool's core workflow – upvoting, downvoting, and iteratively refining candidates – functions as described.

B.1 Navigating the Regex Extension

Regardless of how VS Code is configured, you can always open PICK via the Command Palette:

1. Open the Command Palette with **Cmd+Shift+P** (macOS) or **Ctrl+Shift+P** (Windows/Linux).
2. Type `View: Show PICK Regex Builder` and select the matching command (fig. 1).

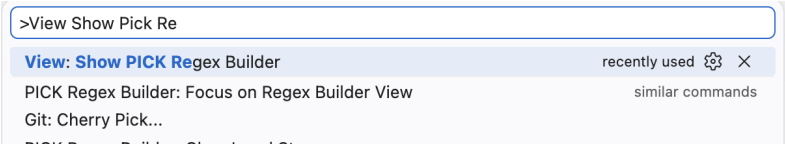

Figure 1 Opening PICK from the VS Code Command Palette.

This will bring the PICK panel into view. From here, select a language model from the dropdown, enter a natural-language description of the regex you want (e.g. *"unix filepaths"*), and use the upvote/downvote controls to iteratively refine the generated candidates. Verify that the UI and functionality resemble those in paper fig. 2.

C Study Data Analysis

The `user-study-raw-data/` folder contains the anonymized raw response data from our Prolific user studies:

- `regex-control.csv`, `regex-with.csv`, `regex-without.csv` – participant responses for the three conditions of the regex study (paper section 4.2).
- `policy-control.csv`, `policy-with.csv`, `policy-without.csv` – participant responses for the three conditions of the access-control study (paper section 4.3).
- `ltl-without.csv` – participant responses for the LTL study (paper section 4.4).

The data analysis conducted for the user studies in paper section 4 can be reproduced locally with Docker. Load the image from the bundled tar file and run it:

```
docker load -i data-analysis-ecoop-artifact.tar
docker run --rm \
  data-analysis-ecoop-artifact:latest
```

The container runs a single analysis script and prints the computed statistics to standard output. This output should be interpreted as a direct replication check against the corresponding results in the paper. The output is presented in three parts, described below with the expected verbatim output for each.

Regex study accuracy (paper table 2). The first part reports per-problem accuracies and pairwise chi-squared tests across the three conditions. You should expect the following:

```
Table 2: Accuracy on regex study
| 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 |
```

Regex experience vs. usefulness (paper table 3). The output also prints a cross-tabulation of mean accuracy by self-reported regex experience and perceived tool usefulness, corresponding to paper table 3:

```
Table 3 (Regex Control): Mean accuracy (%) by Regex experience x Tool usefulness
| Experience | Very useless | Somewhat useless | Neither ... | I'm 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)   |
```

Access control study accuracy (paper table 4). The next part reports per-problem accuracies and pairwise condition comparisons for the policy study. You should expect the following, corresponding to paper table 4:

```
Table 4: Accuracy on access control (policy) study
| 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, .0698  | 0.00, 1.0000 |
| Technology | 46%     | 60%       | 64%          | 1.45, .2293  | 2.59, .1078  | 0.04, .8368  |
```

LTL study (paper section 4.4). The final part evaluates the LTL claims from paper section 4.4. First, Fisher's exact tests comparing PICK participants to those in *P23* and *FM24*:

```
RedOnce:
  PICK vs P23: Fisher p=0.000, OR=6.27
  PICK vs FM24: Fisher p=0.000, OR=6.38
RedOnOff:
  PICK vs P23: Fisher p=0.245, OR=1.64
  PICK vs FM24: Fisher p=0.830, OR=1.18
RedBlue:
  PICK vs P23: Fisher p=0.073, OR=0.47
  PICK vs FM24: Fisher p=0.388, OR=0.64
```

Verify that RedOnce shows $p < 0.001$ with OR ≈ 6.3, matching the paper. Next, non-inferiority tests (Miettinen–Nurminen score test, 10% margin):

```
RedOnce:
  Non-inferiority vs P23: Non-inferior (p=0.000)
  Non-inferiority vs FM24: Non-inferior (p=0.000)
RedOnOff:
  Non-inferiority vs P23: Non-inferior (p=0.005)
  Non-inferiority vs FM24: Inconclusive (p=0.081)
RedBlue:
  Non-inferiority vs P23: Inconclusive (p=0.747)
  Non-inferiority vs FM24: Inconclusive (p=0.489)
```

Finally, the pooled analysis across all three problems:

```
Pooled (all 3 problems):
  PICK: 84/150 = 0.560
  P23: 104/240 = 0.433
  FM24: 59/134 = 0.440

  Pooled NI vs P23: Non-inferior (p=0.000)
  Pooled NI vs FM24: Non-inferior (p=0.000)
```

Verify that pooled non-inferiority is confirmed ($p < 0.001$) against both prior studies, matching the paper's claim.

D Regex Study Application

This section describes how to run the application used for the regex study described in paper section 4.2. The regex study had three conditions: *Control*, *With List*, and *Without List*. Load the image from the bundled tar file and start the container. The `SHOW_CANDIDATES` environment variable controls whether candidates are visible (*With List*) or hidden (*Without List*):

```
docker load -i regex-study-ecoop-artifact.tar

# With candidate list (withList condition)
docker run --rm -p 8080:8080 -e SHOW_CANDIDATES=true \
  regex-study-ecoop-artifact:latest

# Without candidate list (woutList condition)
docker run --rm -p 8080:8080 -e SHOW_CANDIDATES=false \
  regex-study-ecoop-artifact:latest
```

The application will be available at `http://localhost:8080`.

The interaction flow mirrors the regex study procedure described in paper section 4.2.2:

1. Enter the participant identifier on the Prolific PID page (pre-filled in the artifact build for convenience).
2. Review and accept the informed consent page.
3. Complete the screener questions. Select **Yes** for the programming-experience question, then select a regex-experience option indicating at least prior exposure (e.g., seen regexes before, occasional use, or regular use). Selecting **No** or **I'm not familiar with regexes** screens the participant out.
4. Read the instruction page. Participants are asked to classify examples according to the stated specification, rather than an intuitive interpretation of the description.
5. Complete the four-part interface walkthrough.
6. For each regex problem, classify examples until a single candidate remains or all are eliminated; then review and optionally revise classifications before moving to the next problem.
7. If any classification differs from the expected answer, provide a brief free-text explanation.
8. Complete the follow-up questionnaire on perceived tool usefulness, confidence, and technical issues.

This classify-review workflow repeats for all four regex problems.

***Control* condition.** For the regex *Control* condition, we include the original Qualtrics survey as `qualtrics-regex-control.pdf` in the artifact root for reference. The survey flow is:

1. Enter Prolific PID, then the participant must pass the mobile-device check.
2. Review informed consent and proceed only if consent is granted.
3. Complete screener questions on programming experience and regex familiarity.
4. Read task instructions and confirm understanding.
5. Complete four regex-selection tasks (a/b words, time, date, variable names).
6. Complete follow-up questions on perceived usefulness and confidence, then return to Prolific.

E Access-Control Policies Study Application

This section describes how to run the applications used for the access-control study described in paper section 4.3. The ABAC study had three conditions: *Control*, *With List*, and *Without List*. The *With List* and *Without List* conditions use the PICK study application, while the *Control* condition uses a separate application.

PICK application (*With List* and *Without List* conditions). Load the image from the bundled tar file and start the container. The `SHOW_CANDIDATES` environment variable controls whether candidates are visible (*With List*) or hidden (*Without List*):

```
docker load -i policy-study-ecoop-artifact.tar

# With candidate list (withList condition)
docker run --rm -p 8080:8080 -e SHOW_CANDIDATES=true \
  policy-study-ecoop-artifact:latest

# Without candidate list (woutList condition)
docker run --rm -p 8080:8080 -e SHOW_CANDIDATES=false \
  policy-study-ecoop-artifact:latest
```

The application will be available at `http://localhost:8080`.

The interaction flow mirrors the access-control study procedure described in paper section 4.3.2:

1. Enter the participant identifier on the Prolific PID page (pre-filled in the artifact build for convenience).
2. Review and accept the informed consent page.
3. Complete the screener questions. Select **Yes** for the programming-experience question, then confirm familiarity with Boolean logic. Selecting **No** for either question screens the participant out.
4. Read the tutorial page, which introduces access-control policies, explains what subjects, actions, resources, and attributes are, and walks through an example request and its decision (permit or deny).
5. Complete the training questions (3–4 sample classifications with feedback) to ensure understanding of the setting.
6. Complete the interface walkthrough.
7. For each access-control problem, classify request scenarios until a single candidate policy remains or all are eliminated; then review and optionally revise classifications before moving to the next problem.
8. If any classification differs from the expected answer, provide a brief free-text explanation.
9. Complete the follow-up questionnaire on prior access-control experience, perceived tool usefulness, confidence, and technical issues.

This classify-review workflow repeats for all three access-control problems.

***Control* condition.** Load the image from the bundled tar file and start the container:

```
docker load -i policy-control-ecoop-artifact.tar
docker run --rm -it -p 8080:8080 \
  policy-control-ecoop-artifact:latest
```

The application will be available at `http://localhost:8080`.

The *Control* condition does not use PICK. Instead of classifying scenarios, participants select a policy directly from a static list of candidates. The flow is:

1. Steps 1–5 are the same as for the PICK application (identifier, consent, screener, tutorial, and training).
2. For each access-control problem, review the candidate policies and select the one that matches the description, indicate that none is correct, or indicate uncertainty.
3. Complete the same follow-up questionnaire.

F LTL Study Application

This section describes how to run the application used for the LTL study described in paper section 4.4. Load the image from the bundled tar file and start the container:

```
docker load -i ltl-study-ecoop-artifact.tar
docker run --rm -it -p 8080:8080 \
  ltl-study-ecoop-artifact:latest
```

The application will be available at `http://localhost:8080`.

The interaction flow mirrors the LTL study procedure described in paper section 4.4.2:

1. Enter the participant identifier on the Prolific PID page (pre-filled in the artifact build for convenience).
2. Review and accept the informed consent page.
3. Complete the screener questions. Select **Yes** for the programming-experience question, then confirm familiarity with Boolean logic. Selecting **No** for either question screens the participant out.
4. Read the tutorial page, which introduces the domain – an instrument panel with three colored lights (Red, Green, Blue) – gives an example trace, and explains what it means for a trace to be consistent or inconsistent with a temporal description.
5. Complete the training questions (3–4 sample classifications with feedback) to ensure understanding of the setting.
6. Complete the interface walkthrough.
7. For each LTL problem, classify trace scenarios until a single candidate remains or all are eliminated; then review and optionally revise classifications before moving to the next problem.
8. If any classification differs from the expected answer, provide a brief free-text explanation.
9. Complete the follow-up questionnaire on prior LTL experience, familiarity with temporal-reasoning tools, perceived tool usefulness, confidence, and technical issues.

This classify-review workflow repeats for all three LTL problems. Note that, unlike the regex study, candidates are not shown to participants (without-list condition) to avoid requiring participants to read LTL formulae.

G Reusability

The PICK VS Code extension is open-source and available at:

`https://github.com/sidprasad/pick-regex`

Its architecture is built for reuse and adaptation to other domains, with two main layers:

- **Core algorithm** (`pickController.ts`). Manages the candidate elimination loop: tracking candidates, computing vote thresholds, detecting termination, and requesting distinguishing scenarios. This module is independent of any particular formal language.

- **Domain analyzer** (`regexAnalyzer.ts`). Provides three operations: membership testing, semantic equivalence checking, and distinguishing-scenario generation. The core controller delegates to these operations without knowledge of the underlying language.

As described in paper section 2.3, PICK can be used with any formal language that supports closure under complement and intersection (to compute set differences between candidates) and decidable membership with a generative sampling process (to produce concrete scenarios). To instantiate PICK for a new domain, one replaces the domain analyzer with an implementation that provides the three operations above for the target language.