Databending as a Target for Genetic Improvement

GI Workshop at ICSE 2026
Erik M. Fredericks, Byron DeVries, and **Reihaneh Hariri**
Grand Valley State University

Why Creative GI?

- GI usually improves software
- We explore GI for creative output
- Target domain: databending / glitch art
- Question: can evolving code create novel visual behavior?

What is databending?

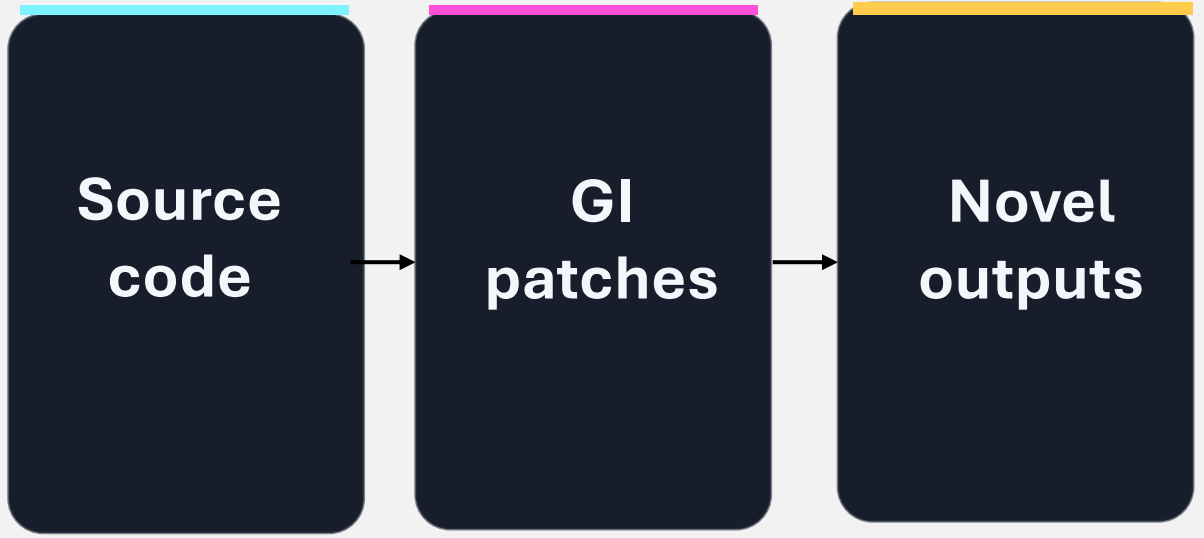
- Intentional corruption for artistic effect
- Warped, distorted, glitched visuals
- We evolve the program, not the image

Original | Test Set | PYGGI-Generated

Databending: Bergstrom & Lotto (2015); Menkman (2011) | Figure: Fredericks et al., GI 2026, Fig. 1

Why is this a GI problem?

- Patch the generator, not the artifact
- Shift from software quality to output novelty
- Opens a new creative GI domain
- Fitness design becomes central

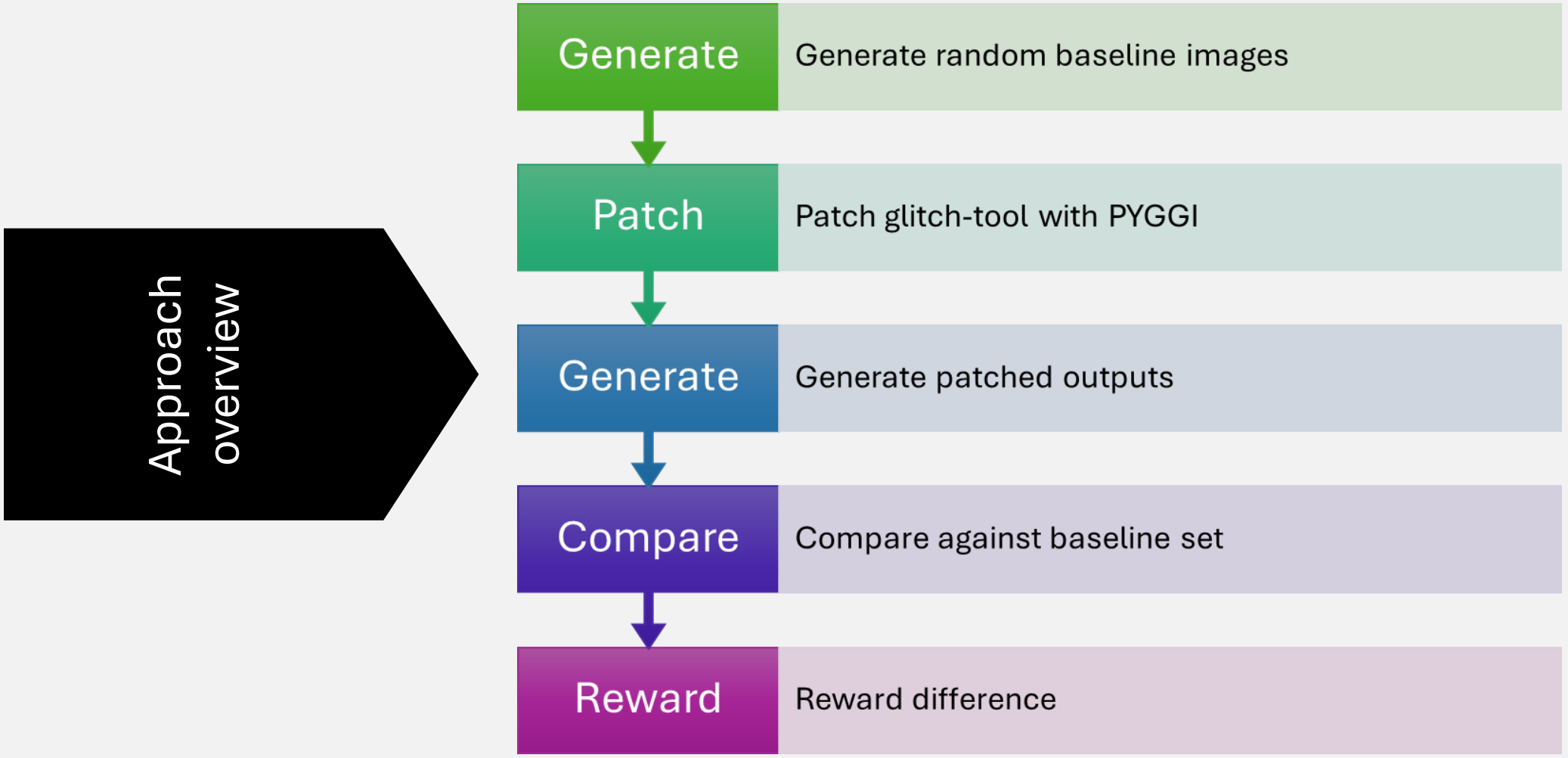


The search acts on software, but the objective lives in the output space.

Why glitch-tool?

- Open-source Python glitching program
- Designed as a proof of concept
- Not intended to be maintained or production-ready
- Good sandbox for exploratory GI

How the Search Works



What Was Randomized vs. Evolved?

RANDOMIZED

- glitch mode
- number of changes
- bytes changed
- bytes repeated

EVOLVED

Source code only

Parameter tuning was intentionally left outside the search so the experiment measured code evolution, not configuration optimization.

Control matters: novelty should come from patches, not from simply tuning parameters.

Initial Fitness Design

- Average hash for each image
- Hamming distance to test-set images
- Fitness = image difference + patch-size term

Hamming distance:

$$h_{diff} = |avg\_hash(new\_image) - avg\_hash(test\_image)|$$

A lightweight proof-of-concept fitness signal

Experimental Setup

25
replicates

Tree
mode

Tabu
search

10
iterations

1000
epochs

Data

5000 generated test images
2094 valid images
Average-hash comparison

Interpretation

Over half of the initial images were too corrupted to use in the image-hashing fitness calculation.

That is not just noise—it is part of the challenge of this search space.

Example patches

- Variable and function transplants
- Many “best” patches were trivial
- Suggests limited movement in the search patch space
- Possible local optima or constrained

Representative patch excerpt

```
    iteration , bytesToChange , seed )
    def changeBytes( byteList , \
      bytesToChange ):[40/1848]
        global args
        pos = random.randint(0, \
          len( byteList ) - bytesToChange )
        chunk = [random.randint(0, 255) \
            for i in range(bytesToChange )]
        byteList[pos:pos + bytesToChange ] \
            = chunk
        return byteList
writeFile(newByteList , fileNum , iteration ,\
    bytesToChange , seed )
```

Best visual outputs

- PYGGI scored slightly higher
- Visual difference appears mainly in color palette

Best test-set image

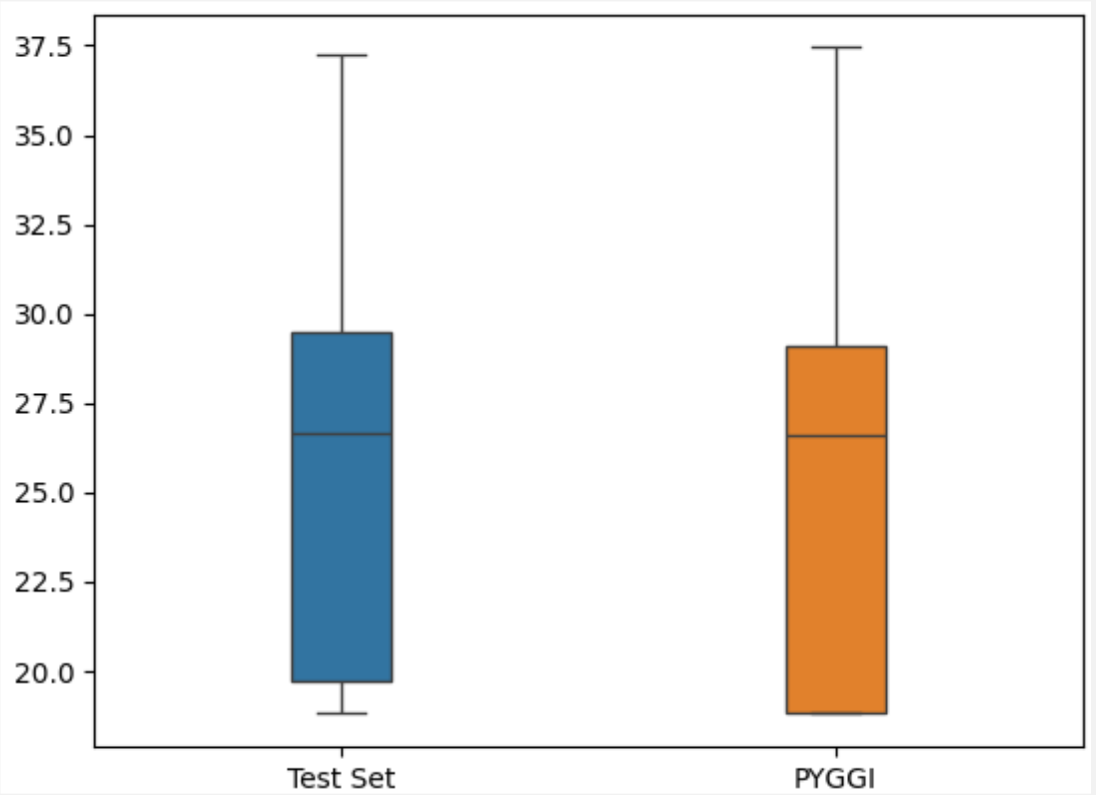
Test set image

Best PYGGI-generated image

PYGGI-image

Result: Interesting, but Not Beyond Random

- No significant difference overall
- PYGGI performs roughly on par with random generation
- Feasible, but not yet clearly superior



Average Hamming distances comparisons between test set and PYGGI-generated images.

What Did We Learn?

Feasible new GI domain

Databending can be instrumented as a genuine GI problem.

Objective design matters

The core difficulty is defining novelty meaningfully.

Search may plateau

Trivial patches suggest local optima or a constrained patch space.

Creative GI needs stronger novelty signals than simple output difference alone.

Where Next?

Better novelty metrics

perceptual hash
wavelet hash
histogram / pixel / ML methods

Novelty search

Move beyond a single scalar objective toward explicit diversity-seeking behavior.

Compare with GenerativeGI

Position the work against related creative GI approaches.

Other creative media

Extend beyond this tool to other generators, images, and video.

This is where the project becomes most interesting for the GI community.

Takeaway

Databending is a feasible new GI domain,
and novelty-aware fitness design is the central open challenge.

Acknowledgements

We gratefully acknowledge the support of **Grand Valley State University**.

Selected References

- An, G., Blot, A., Petke, J., & Yoo, S. (2019).
  PyGGI 2.0: Language Independent Genetic Improvement Framework.
  Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019), 1100–1104.
- Løfgren, T. (2019).
  glitch-tool GitHub Repository.
- Buchner, J. (2025).
  ImageHash GitHub Repository.
- Fredericks, E. M., Moore, J. M., & Diller, A. C. (2024).
  GenerativeGI: creating generative art with genetic improvement.
  Automated Software Engineering, 31(1), 23.
- Fredericks, E. M., Diller, A. C., & Moore, J. M. (2023).
  Generative Art via Grammatical Evolution.
  Proceedings of the 12th International Workshop on Genetic Improvement.
- Lehman, J., & Stanley, K. O. (2011).
  Novelty search and the problem with objectives.
  In *Genetic Programming Theory and Practice IX*, 37–56.
- Blot, A., & Petke, J. (2021).
  Empirical Comparison of Search Heuristics for Genetic Improvement of Software.
  IEEE Transactions on Evolutionary Computation, 25(5), 1001–1011.