93% of Paint Splatters are Valid Perl Programs

Colin McMillen and Tim Toady
twitter.com/mcmillen & famicol.in/sigbovik

Abstract

In this paper, we aim to answer a long-standing[1] open problem in the programming languages community: *is it possible to smear paint on the wall without creating valid Perl?*

We answer this question in the affirmative: it **is** possible to smear paint on the wall without creating a valid Perl program. We employ an empirical approach which finds that merely 93% of paint splatters parse as valid Perl. We analyze the properties of paint-splatter Perl programs, and present seven examples of paint splatters which are *not* valid Perl programs.

Background

In a February 2019 Twitter conversation, Adrienne Porter Felt expressed a desire for her kid to smear paint on the wall instead of learning vocational skills such as computer programming [1]. In response, Jake Archibald posed the question which forms the basis of this research: "is it possible to smear paint on the wall without creating valid Perl?" [2]

While many PL researchers have (often derogatory) folk beliefs about the Perl programming language, the language itself has not been the subject of much formal academic inquiry. One exception is Jeffrey Kegler's proof that the Perl programming language is undecidable [3], often summarized with the maxim "only `perl` can parse Perl".

Another maxim of the Perl community is "There Is More Than One Way to Do It."[2] Despite the popularity of this maxim in the Perl community, the authors are not aware of any professional Perl engineers whose development practices involve smearing paint on walls.

We thus believe that we are the first researchers in academia or industry to directly address the question of whether paint splatters are valid Perl programs.

Experimental Setup

Our approach to answering this question is an empirical one. Given an input image, we run optical character recognition (OCR) software on that image to extract candidate text. As previously mentioned, the question "is this string valid Perl?" is theoretically undecidable; we therefore fed the extracted text into the `perl` executable (version `5.26.1`) to check whether the OCR'd string corresponded to a valid Perl program.

We are not aware of the existence of any standard paint-splatter datasets in the object recognition or OCR communities. Also, ImageNet's website was down on

[1] By "long-standing", we mean "for roughly a month or so".

[2] Often pronounced "Tim Toady"; hence the name of the fictional second author of this paper. (Like many researchers, we collaborate with other authors primarily so that our use of the royal "we" doesn't come across as pretentious.)

the day that we decided to perform this research. We therefore paid an unemployed person[3] to download 100 examples of paint-splatter artwork by searching Pinterest using the query `"paint splatter wallpaper"`.

We manually filtered out all images with any form of overlaid or watermarked synthetic text, because the Perl program (?) `"iStock by Getty Images"` is not particularly interesting.

The resulting 100 images are shown in Figure 1, and are also available online as supplementary material to this paper (see the Addendix).

Figure 1. Paint-splatter image dataset.

[3] The first author.

To perform OCR on the input images, we used the Tesseract OCR engine (version `4.0.0-beta.1`) [4]. Tesseract is an open-source OCR library that provides two separate algorithms for optical character recognition: a "legacy" engine that uses traditional OCR techniques, and a newer engine based on LSTM models. Tesseract also provides a third OCR engine which somehow combines the two. Unfortunately, the documentation does not describe this mode in detail, but we chose to make use of it anyways.

Additionally, Tesseract provides multiple algorithms for performing page segmentation. For example, page segmentation mode #4 assumes that the page consists of a single (possibly multiline) column of text; mode #7 treats the image as a single line of text.

It is possible to configure Tesseract's legacy OCR engine with a list of allowed characters expected to be in the input alphabet. Since Perl programs mostly consist of the printable subset of ASCII, we restricted the OCR engine alphabet to ASCII characters in the range [32, 127). Tesseract's newer LSTM-based engine doesn't appear support this sort of configuration.[4] We also disabled features related to language modeling where possible (for example, penalties for "words" that are not found in the English dictionary).

It was unclear to the authors which OCR engine and page segmentation modes would best correspond to "splats of paint

[4] Apparently in the brave new world of neural nets, Anything Goes™ (except for the ability to configure things.)

that might parse as valid Perl programs". Therefore, for each image, we tried all combinations of the 3 OCR engines and 4 of the 14 different page segmentation modes (modes #4, #7, #8, and #9) to determine which configuration was the most fruitful for producing a valid Perl program out of that specific image.

This "try multiple algorithms until one of them happens to work" approach is profoundly unethical — especially since we don't have separate training, test, and validation sets — but at least we're being honest about what we're doing, instead of inventing a fancy-but-obfuscatory technical term like "ensemble methods" or "hyperparameter tuning".

Results

The main result of this paper is that 93 of 100 images in our dataset successfully parsed as valid Perl programs under at least one combination of Tesseract OCR engine & page segmentation mode. (It is worth explicitly noting that we only considered non-empty Perl programs as successes.)

The most fruitful single combination of parameters is provided by the LSTM engine using page segmentation mode 9, which successfully produces valid Perl programs out of 55% of paint splatters.

The pure LSTM approach was the most successful of the three OCR engines, with 74% of input images successfully parsing as Perl under at least one of the four page segmentation modes. The legacy OCR engine succeeded in finding valid Perl programs on 62% of images, while the combination legacy+LSTM engine succeeded merely 40% of the time. It is unclear why the combination of the two OCR algorithms would be significantly worse at recognizing valid Perl programs than each OCR algorithm on its own.

Discussion & Analysis

While we successfully found valid Perl programs in 93 of 100 input images, all of these programs are uninteresting; they don't actually seem to *do* anything when executed. However, some of them do evaluate to a value, which is then discarded without being displayed. [5]

We therefore passed each valid program into Perl's `eval()` function and printed out the result of evaluating it. Many of these are simple integer literals. Figure 2 shows an input image which is read by OCR as the string "`35`", which evaluates to the number `35` when parsed by Perl.

Figure 2. If you squint, you can see the number `35`.

Figure −3 shows a somewhat more interesting case. This input image is read by OCR as the string "- 3", which evaluates to the number −3 when parsed by Perl.

Figure −3. −3.

In all, the dataset contains 20 images which can be parsed as numeric literals. An interesting thing about these images is that they're not particularly Perl-specific; for example, the program shown in Figure −3 also evaluates as the number −3 in Python (and presumably several other programming languages.)

−3 is the smallest number resulting from the output of our valid Perl programs; the program which evaluates to the largest number is shown in Figure 4.

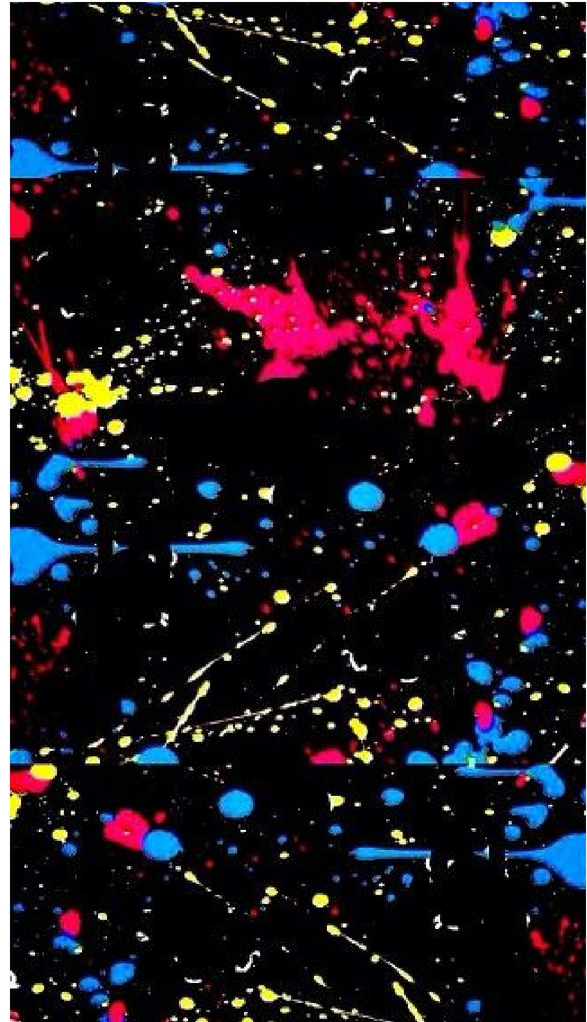

Figure 4. The valid Perl program "225252", which evaluates to 225252.

The remainder of the valid Perl programs are mostly valid due to a Perl language feature that is *not* commonly present in most programming languages. Namely, Perl has a feature called "unquoted strings", in which a sequence of alphanumeric characters by itself is parsed as though it were a quoted string. As an example, Figure 5 is read by OCR as the text `ME`, which evaluates to the string "`ME`" even though the `ME` isn't quoted. This would result in a syntax error in most other programming languages.

Figure 5. It `ME`.

Figure 6 represents the string "`gggijgziifiiffif`", which by pure coincidence happens to accurately represent the authors' verbal reaction upon learning that "unquoted strings" were a feature intentionally included in the Perl language.[5]

Figure 6. `gggijgziifiiffif`!

[5] This feature *does* enable a neat quine: the Perl program "`Illegal division by zero at /tmp/quine.pl line 1.`", when saved in the appropriate location, outputs "`Illegal division by zero at /tmp/quine.pl line 1.`" The reason for this behavior is left as an exercise for the reader.

(To be fair to Perl, when `perl` is run with the `-w` flag to enable warnings, it does helpfully inform the user that at some point in the future, the Perl developers will most likely pick `gggijgziifiiffif` as a new reserved word:

```
Unquoted string
"gggijgziifiiffif" may clash
with future reserved word at -
line 1.)
```

Another interesting case is presented in Figure 7. This image represents the source code "`;`" which is non-empty, but which is just a statement separator that does not evaluate to anything.

Figure 7. A statement of no purpose.

The Rogue's Gallery

At this point, we would not blame the reader for sympathizing with Jake Archibald's conjecture that *all* paint splatters are in fact valid Perl programs. Perhaps Adrienne's kid is doomed to accidentally write valid Perl even when just trying to smear some paint around at random. Here we finally present some counterexamples: the seven images in our dataset which do not, under any OCR interpretation, parse as valid Perl.

Figure 8 presents a splatter which is read by OCR as any of the following strings:

- `fifi;%:'i1i:`
- `.%f:`
- `&`
- `i;%:';;:`

Surprisingly, none of these strings represent valid Perl programs.

Figure 8. Not valid Perl. Obviously.

For completeness, we present the other six such images as Figure 9.

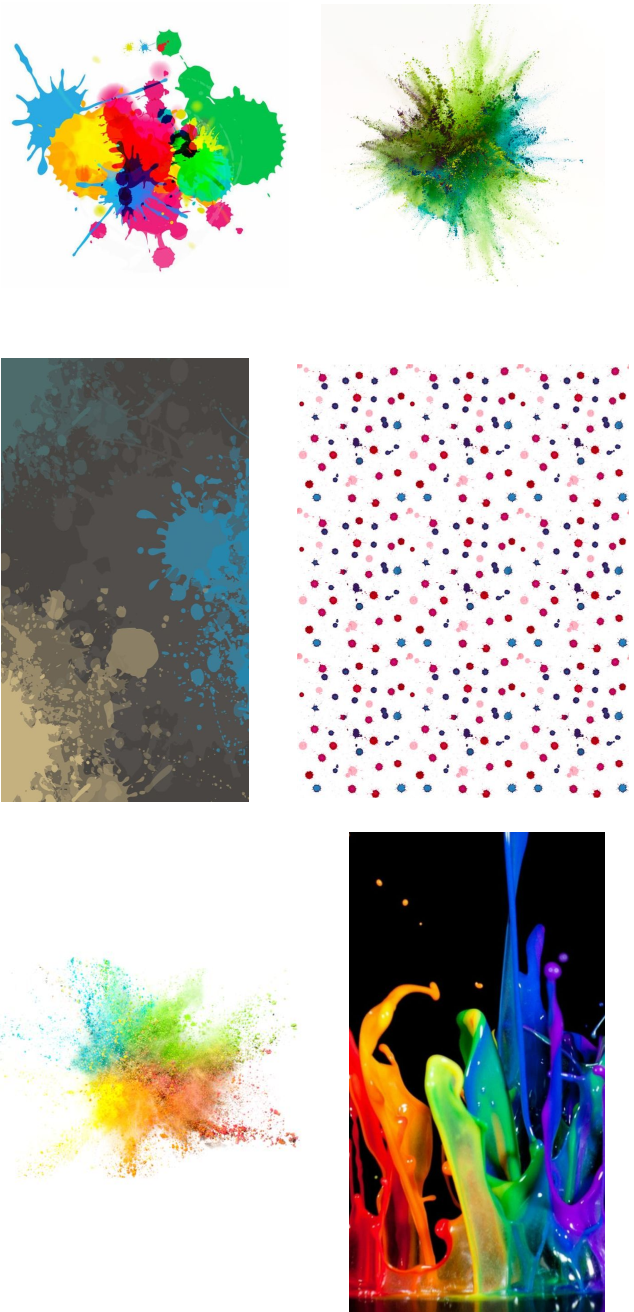

Figure 9. Finally, we are free of the tyranny of accidentally writing valid Perl programs. These are the sorts of paint splatters one might want one's child to produce when they're just having fun.

Woomy?!?

Fans of the *Splatoon* video game series will naturally be wondering whether the "splats" in Nintendo's official game artwork are also valid Perl programs. Our preliminary answer is *yes*: the image in Figure 10, downloaded from Nintendo's official Splatoon website [6], successfully parses as a valid Perl program.

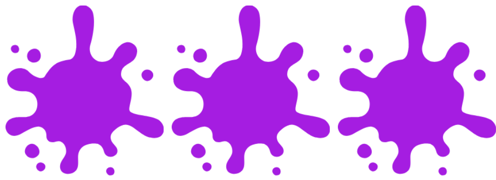

Figure 10. Splat splat splat… woomy?

Given Nintendo's family-friendly image, the authors were surprised to find out that the source code that results from OCR'ing this image is somewhat NSFW.[6] We thus we elide it here, for the sake of SIGBOVIK's younger readers.

Future Work

While the results presented in this paper are novel and important, they only begin to break ground on what could be a very fruitful area of further research.

The dataset used in this paper is a relatively small dataset of only 100 paint-splatter images. It would be good to confirm these results on a larger dataset, and with a greater variety of images. Perhaps next time ImageNet won't be down.

We also noticed far too late that while the original question referred to paint *smears*, we elected to search Pinterest only for paint *splatters*. It is unclear at whether these results would change significantly for paint splatters vs. paint smears.

Similarly, our choice to select images from Pinterest ensured that they were reasonably high-quality paint splatters, as at least one Pinterest user had chosen to "pin" that image as something worth saving for later. It would be worth investigating whether amateurish, lower-quality paint splatters — such as those produced by a young child — are less likely to be parsed as valid Perl programs.

After downloading the 100 images used in our dataset, Pinterest somehow inferred that the authors of this paper might be interested in images of "swimwear trends". We have not yet investigated whether 2019's latest swimwear trends are more or less likely to parse as valid Perl programs.

[6] Really, it's Perl itself that's most unsafe for work.

Addendix

The source code for the research presented in this paper, as well as the full dataset of 100 paint-splatter images & the result of evaluating each, will soon be available at http://famicol.in/sigbovik.

References

[1] Adrienne Porter Felt. https://twitter.com/__apf__/status/1095698777300586496.

[2] Jake Archibald. https://twitter.com/jaffathecake/status/1095706032448393217.

[3] Jeffrey Kegler. “Perl Is Undecidable”. The Perl Review, Volume 5, Issue 0, Fall 2008, pp. 7-11. Available online at http://www.jeffreykegler.com/Home/perl-and-undecidability.

[4] Tesseract Open Source OCR Engine. https://github.com/tesseract-ocr/tesseract

[5] `Patrician|Away` and `bovril`, personal correspondence. Recorded at http://bash.org/?240849.

[6] Splatoon 2 amiibo™ home page. https://splatoon.nintendo.com/amiibo/.