PATHOS, VECTORS, MIST

HANNES BAJOHR

In the realm of computation, we often speak of "syntax" and "grammar," of "expressions" and "statements," but also of "elegant solutions" and "aesthetic code." Programming languages—from Assembly to Python, from SQL to Rust—serve as bridges between human thought and machine execution. But beyond their functional capacity, beyond their ability to process and transform, their status as language is problematic, despite the use of such linguistic and artistic vocabulary. This point comes starkly into view once we ask a strangely conservative question: Are programming languages, are codes open to emotional and affective aspects? To put it bluntly, can pathos (πάθος, suffering/emotion/affect) be programmed? I shall try to approach this issue from two angles: first, by examining the relationship between pathos and programming languages and, second, by turning toward pathos and artificial intelligence.

1. Pathos and Programming

The first problem emerges from a fundamental misunderstanding about the nature of code itself: Programming languages are not *languages*. You cannot speak German *and* C++ like you can speak English or French; indeed, you cannot *speak* C++ at all.[1]

1. So writes computer scientist Edsger W. Dijkstra: "The introduction of the term 'language' in connection with notation techniques for programs has been a mixed blessing. . . . [T]he analogy with (now so-called!) 'natural languages' has also been very misleading, because natural languages, non-formalized as they are, derive both their

Programming languages describe calculations for processing information. Or, according to an old shorthand from computer science: "Programs = Algorithms + Data Structures."[2] This means that there is a fundamental bifurcation within digital language itself. On the one hand, there are the rules for data processing (the algorithm), which are not written in a natural language; on the other, there are the data structures, which *can* represent a natural language—a poem encoded as a plain text file, for instance, the encoding of which is, of course, no longer a natural language—but do not have to; anything is possible, from a database to an image file or a vector graphic. Algorithms and data structures are not only conceptually distinct but also represent separate domains of meaning. This is not to say that both levels of language are strictly separate; in practice they can be deeply intertwined: Certain algorithms work best with specific data structures, and the choice of data structure can fundamentally affect the algorithm's efficiency and implementation. But pathos seems to escape that connection entirely.

The reason for this is that algorithms expressed in code are purely imperative, instructional. They have no connotative, referential, or truly pragmatic dimension. In this, they take the conventionality of language—the fact that the system of language is symbolic rather than iconic or indexical—to an extreme. All similarities codes might display to words in natural languages are purely a convenience for human users to learn, remember, and operate them. Although code languages are designed for readability, this property is decoupled from the instructions that govern their execution. Programming guru Donald Knuth is therefore often quoted to

---

weakness and their power from their vagueness and imprecision." Edsger W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, 1976), 8. Interestingly, Dijkstra's complaint about the term "language" is aimed at the difference in formal exactness, not the difference in meaning-making power. In fact, "semantics" for Dijkstra, and for computer science in general, signifies the mathematically rigorous definition of how a program transforms inputs into outputs, how it arrives at its "post-conditions" (output) by systematically deriving its weakest "preconditions" (input). Dijkstra, chap 3. This is very much in line with what one might call the basic preliminary decision as to meaning in computer science: namely, that "semantic aspects of communication are irrelevant to the engineering problem." Claude E. Shannon, "The Mathematical Theory of Communication," in Claude E. Shannon and Warren Weaver, *The Mathematical Theory of Communication* (University of Illinois Press, 1964), 31.

2. Niklaus Wirth, *Algorithms + Data Structures = Programs* (Prentice-Hall, 1976).

have written, "Programming is the art of telling another human being what one wants the computer to do."[3] Anything that goes beyond linking these human-readable instructions with effects in the processor and memory of the computer is irrelevant to their function as a programming language. Codes are, as the philosopher Charles Taylor put it, the most extreme outcrop of a "designative" theory of language, a language that is absolutely devoid of language as constitutive of a world-relation; they are, in their most extreme case, "representation without expression."[4]

This makes the answer to our question simple. If there is programmed pathos to be found at all, then probably only so on the part of the data structure, not the algorithm. But this is trivial—the *processing* of a text would then be completely separate from the *content* of this text. Operating on data structures has no traffic with their meaning. One could read out a spreadsheet instead of a poem to the same effect.

However, this somewhat strict position presupposes that the code is *only* read or allowed to be read as code, that is, only in its functionality, and that ignores everything that plays no role in the execution, such as the fact that certain commands in programming languages can, of course, have real-world connotations: for example, "kill," which terminates a process, "sleep" for a pause in the computing cycle, and so on. The same goes for

3. This is an abbreviated aphorism that has been circulating for years and is quoted, for example, in Selma Tekir, "Reading CS Classics," *Communications of the ACM* 55, no. 4 (2012): 33. This refers to a somewhat less pithy line in Knuth's "Literate Programming": "Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do." Donald E. Knuth, "Literate Programming," *Computer Journal* 27, no. 2 (1984): 97.

4. Charles Taylor, "Theories of Meaning," in *Human Agency and Language: Philosophical Papers I* (Cambridge University Press, 1985), 267. Of course, code can have a rhetorical, aesthetic, and even ideological function, as scholars such as Wendy Hui Kyong Chun and Matthew Fuller have argued, but such an interpretation always needs to be situated in a double perspective: code must be read both as instruction and as inscription. Its rhetorical or poetic force does not arise from its executability alone but from how it is embedded in social, cultural, and medial contexts. The very fact that code can be commented, stylized, or arranged for human readers—even when doing so has no impact on its execution—attests to its legibility beyond the machine. Yet this surplus meaning is always parasitic on the gap between the symbolic and the operational, between what the computer does and what we see it do.

variables freely chosen by the programmer, which can carry meaning for their readers if not the machine.[5]

But code does not have to be read *as* code. Code poetry is a historically influential literary example for an approach that plays with the connotative and aesthetic surplus of the functionally "meaningless" code. It undermines the difference between human and machine readability by mixing both types of text without eliminating either of their characteristics entirely. One way it does so are comments. Comments are lines in the code, usually preceded by some special characters, which are not executed. They are addressed not to the machine but to the human actors, to the coders themselves or future readers, and are thus neither executable nor entirely non-functional. Their execution is simply not part of the machine alone but the secondary semiosis of the human-machine context in which they appear.

Take, as an example for this entanglement, the poem "three last words" by Lillian-Yvonne Bertram included in their 2019 poetry collection *Travesty Generator*. In the book, the title of the poem is stylized as: "#/usr/bin/python/three_last_words." This looks like a standard first line in a Python script specifying the interpreter (the program executing the code) used to execute the file (usually "#/usr/bin/python").

The poem itself is written in Python. On the first page, a function is defined, which rearranges an input string (a chunk of text) and appends it back to itself; the execution of this function is listed on the following pages. The function, called "permutations," processes an input, which is called "elements" here.

```
def permutations(elements):

#the

    if len(elements) == 0:

#the knife

        yield elements
```

5. Critical code studies turns this approach into an entire research program by examining the cultural connotations of code. See Mark C. Marino, *Critical Code Studies* (MIT Press, 2020).

```
        #the knife they
        else:
    #the knife they hung
            for result in permutations(elements[1:]):
                #the knife they hung him
                for i in range(len(elements)):
                    #the knife they hung him on
                    yield result[:i] + elements[0:1] + result

                                #the knife they hung him on
                                #was a legal trinket
```

This is only the abstract definition of the function. Nothing is output here yet; only the rule is introduced according to which something can be processed. However, the text on this page—code as instructional logic, a denotative rule without immediate expression—is interrupted by comments. These begin with "#" and are marked as non-executable for the interpreter; it simply ignores them as if they were not there. Here, natural language text is forced into the executable script lines as an externally meaningful, yet internally meaningless, *annotation*. In other words, the only meaningful lines, if you deny code any expressive function, are precisely those that mean nothing to the computer. Pathos—if that is what it is—would lie completely *beyond* the program here, and fundamentally so.

But that is perhaps too simple. The question was about whether pathos can be programmed. Later in this code poem, pathos is conveyed not only by the commentary but also by the code itself—not in its output, but in the way it is *processed*, in the logic that the program applies and which is based on the architecture and limits of the computer.

On the next page of the poem, the abstractly defined function is called for the first time; instead of "elements," which was only a placeholder in the definition, a character string is now actually processed: "I." Since a single letter cannot be permuted, the output remains "I" (an all-too stable identity?). This is followed by "can't." As the maximum number of permutations of five characters in the logic of the code is 5!=120, the letters are jumbled up accordingly. The script proceeds to do this operation with the word "breathe" (7!=5040) and, finally, processes the whole sentence: "I can't breathe." Understood as a string with 15 letters (including spaces), this yields 1,307,674,368,000 outputs—so many that the program crashes and, significantly, throws a "memory error."[6]

print (list(combinations("**I** can't breathe")))[7]

MemoryError Traceback (most recent call last)

<ipython-input-6-9f1f016de5c5> in <module>()

----> 1 **print** (list(combinations("**I** can't breathe")))

MemoryError:

"#/usr/bin/python/three_last_words" is dedicated to Eric Garner, who died in the chokehold of a New York police officer on June 17, 2014; his last words were "I can't breathe." The comments that Bertram inserted into the code refer in turn to Freddie Gray, who died in police custody when he was stopped on suspicion of carrying an illegal knife, the "legal trinket" the comments reference.

6. The resulting number is likely even higher: Unlike standard permutations that create separate, equal-length arrangements—and unlike the script by author and programmer Nick Montfort Bertram used as their model—this version concatenates (joins together) each new arrangement with the previous results, creating progressively longer strings, see Zach Whalen, "'Any Means Necessary to Refuse Erasure by Algorithm': Lillian-Yvonne Bertram's *Travesty Generator*," *DHQ: Digital Humanities Quarterly* 17, no. 2 (2023): ¶ 40. This means the computer must store not just over a trillion separate 15-character arrangements, but increasingly lengthy strings that grow without limit—turning an already impossible computational task into something even more mathematically extreme.

7. That Bertram's poem in fact calls a function called "combinations" and not the earlier defined "permutations" could itself be a source for interpretation. I chose to simply see it as a typo or an artifact of an earlier version, since an undefined function would return a name error instead of a memory error.

The "memory error," the impossibility of comprehending the Black suffering caused by police violence, of "processing" it and remembering it appropriately, of memorializing it—that would be the affective element of the code, a kind of performative code pathos that remains solely on the operational level and *only* comes about through the denotative function of a system that deals with symbols and their operational limits. As Zach Whalen writes in his detailed analysis of the poem, "Bertram's software presents the volatility of memory as the entanglement of poetic representation, police brutality, and technical limitation, and by setting it up to fail, Bertram invites us to reenact the trauma of that brutality and interpret its technical and political consequences."[8]

What fails here is not comprehension or understanding or meaning on the part of the *system*; the mere processing of symbols fails there, beyond their meaning *for us*. All the meaning that comes back in here is always already outside of the system and is the result of the framing, the staging that Bertram performs and which we first must interpret. The philosopher Maurizio Lazzarato refers to precisely this circumstance when he writes, "Sign machines operate 'prior' and 'next' to signification, producing a 'sense without meaning,' an 'operational sense.'"[9] Such an operational sense seems to come into play masterfully in Bertram's work. Yet it is not the only one there is.

2. Pathos and AI

Despite all of this, code *in itself* hardly seems suitable for producing pathos, affect, emotion. In Bertram's *Travesty Generator*, the pathos function arises from embedding a Python script in an interpretative context that required an external meta-reflection on the logic of the machine. But are there types of processing that themselves are capable of what Aby Warburg has called *Pathosformel*, both embodied and iterable, a symbolic expression charged with intensity that survives through repetition and transformation?[10] I don't

8. Whalen, "'Any Means Necessary'": ¶ 48.

9. Maurizio Lazzarato, *Signs and Machines: Capitalism and the Production of Subjectivity* (MIT Press, 2014), 24.

10. Aby Warburg, *The Renewal of Pagan Antiquity: Contributions to the Cultural History of the European Renaissance* (Getty Research Institute for the History of Art and the Humanities, 1999), 15.

think this is the case for classical programming languages. In them, there is more of a kind of necessary coldness exemplified in the case of permutation, the combination of all elements of a corpus. It is, of course, quite possible that repetition itself can give rise to an affective pathos, but examples of this beyond devices like anaphora are rare. Absurdity and humor, however, are much easier to generate: The incongruity that is decisive for them is quickly created by a recombination of a set of elements (as in the game Mad Libs, template sentences to be filled with freely chosen words). Pathos is perhaps also a kind of incongruity—between a measure and its excess—but one that requires further, more comprehensive structures of meaning that must be able to operate in the background and define the boundary at which affective excess occurs. *Pathos without reference* would be the only thing a computer could produce, and the absence of reference is possibly more comical than sublime.

Machine learning may present other possibilities here because it has a fundamentally different way of operating with digital language than the classic code used by Bertram.[11] It is often disparagingly referred to as "statistics on steroids"—just calculation—while the discourse of the last few years has condemned it as a particularly inhuman way of writing.[12] But two things are important here.

First, the latent space produced by a language model—encoding statistical correlations between tokens (partial words) that predict which tokens are most likely to follow others based on vast training data—is very much unlike algorithm and data structure.[13] It is no longer feasible to translate the model's behavior into discrete, human-readable rule-based steps. This does not merely imply a "lack of transparency" or the black-box problem of

11. On the difference between classical programming and machine learning—or "sequential" and "connectionist" paradigms—see Hannes Bajohr, "Algorithmic Empathy: Toward a Critique of Aesthetic AI," *Configurations* 30, no. 2 (2022): 203–31. For an excellent technical and semiotic introduction to LLMs, see Paul Kockelman, *Last Words: Large Language Models and the AI Apocalypse* (Prickly Paradigm Press, 2024).

12. Just to cite the most prominent example, see Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell, "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?," in *FAccT '21: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (Association for Computing Machinery, 2021), 610–23.

13. See Antonio Somaini, "A Theory of Latent Spaces," in *The World Through AI: Exploring Latent Spaces*, ed. Antonio Somaini (JBE Books, 2025).

AI, which refers to the inability to fully understand how a given result was produced. Rather, it primarily means that AI models function differently enough that the limitations of classical code may not be the same as those of AI models, or at least that they might be different.

Second, although language in AI is also just a material that is processed, the difference to the computation of yore comes to the fore most starkly in the way this language is encoded—specifically, as multidimensional vector embeddings, as complex numerical lists that preserve relationships with all other word vectors. Elements of language here, as in Ferdinand de Saussure's linguistic structuralism[14] or W. V. O. Quine's semantic holism, are in fact defined solely by their position within the totality of its system. In each vector, the whole is present and in turn the whole only consists of the relationship of elements to other vectors.[15] And because not only relations of words to words are encoded here, but also relations *of relations*—second-degree properties, latent vectors—such models can also include those relations that are difficult or impossible to grasp via mere rules.

Such emergent properties imply categories that are not encoded at the word level. A by now canonical example of vector arithmetic looks like this: $\overrightarrow{king} - \overrightarrow{man} + \overrightarrow{woman} \approx \overrightarrow{queen}$.[16] Read: If you subtract the vector for man from king and add the vector for woman, you will get something like the vector for queen. Gender appears here—without having been stated expressly—as a latent result of complex relations of vectors. (This is also a source of bias as racist, sexist, ableist, and other ideologies can remain latent yet operative in

14. See Juan Luis Gastaldi, "Why Can Computers Understand Natural Language? The Structuralist Image of Language Behind Word Embeddings," *Philosophy & Technology* 34, no. 1 (2021): 149–214; and Leif Weatherby, *Language Machines: Cultural AI and the End of Remainder Humanism* (University of Minnesota Press, 2025).

15. Nicely explained in Michael Gavin, "Is There a Text in My Data? (Part 1): On Counting Words," *Journal of Cultural Analytics* 5, no. 1 (2020); for a good introduction, see Mercedes Bunz, "Thinking Through Generated Writing," in *Thinking* with *AI: Machine Learning the Humanities*, ed. Hannes Bajohr (Open Humanities Press, 2025), 84–111.

16. See Tomáš Mikolov, Wen-Tau Yih, and Geoffrey Zweig, "Linguistic Regularities in Continuous Space Word Representations," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (ACL, 2013). I develop this argument in more detail in Hannes Bajohr, "Dumb Meaning: Machine Learning and Artificial Semantics," *IMAGE* 37, no. 1 (2023): 58–70.

the same way.) If language models can make implicit knowledge explicit, and make a social aspect of language like gender appear, why not also other social dimensions of language? Why not pathos? If classical computation was built on the explicit exclusion of semantic meaning, then the different operational logic of LLMs might indeed represent a departure from these foundational constraints. Could affect itself—however imperfectly—take on a geometric form in the latent space of a language model?

To be sure, there are good reasons to believe that language models have no real-world relation and cannot "feel" pathos. It is clear that, apart from syntax, they lack almost everything else that belongs to language proper: semantics and pragmatics, object reference, and the pre-linguistic tacit knowledge that characterizes every meaningful world experience.[17] Despite the ever more impressive feats of language models, one should therefore not be fooled by them. Understanding, meaning, and so on exist in their full sense, most likely only ever on the user side, as projection or as wish fulfillment.

But if we abandon "meaning" as an intentional category that is connected to communicative intent, world models, or models of other minds, and think of it solely as a relationship of relations within a *latent space of language*[18]—in which concepts, styles, moods, "vibes"[19] that are not explicitly encoded can also emerge implicitly—then I do not see why something like pathos could not be extracted in the same way as gender.

Meaning should then be understood as an interference effect of data, as something that simply arises or comes to light without an underlying consciousness. Language—and pathos—would be cloaked in a strong materialism. Lydia Liu uses the apt metaphor of the telephone system, which does not care what runs through its cables. She writes, "[M]aterial language [. . .] operates in the manner of a telephone exchange system or a cybernetic machine that runs automatically regardless of what happens to pass down

17. See Dimitri Coelho Mollo and Raphaël Millière, "The Vector Grounding Problem," arXiv, April 4, 2023, https://doi.org/10.48550/arXiv.2304.01481.

18. Hence the discussion of LLMs as a kind of "fully activated formalism." Matthew G. Kirschenbaum, "Spec Acts: Reading Form in Recurrent Neural Networks," *ELH* 88, no. 2 (2021): 378.

19. See Peli Grietzer, "A Theory of Vibe," in Bajohr, *Thinking with AI*, 20–32.

its wires."[20] Likewise, AI could only produce "dumb" pathos,[21] behind the machine's back.

From this perspective, pathos would not be "programmable," but only "extractable." Maybe in a satisfying but in the end meaningless confirmation of those who decry the inhuman language of language models, pathos would emerge as a complex statistical dependency from sufficiently large corpora, just as mist rises from putrid marshes, as the evaporation of dead matter.

20. Lydia H. Liu, *The Freudian Robot: Digital Media and the Future of the Unconscious* (University of Chicago Press, 2011), 189–90.

21. See Bajohr, "Dumb Meaning."