CHAPTER 11

Generated Literature

Nick Montfort and Judy Heflin

11.1 Introduction

Literature has been generated by computers since the beginning of the 1950s. We mean "computer" in the usual sense it is used today: a general-purpose, programmable, electronic digital machine that performs symbol manipulation automatically. We are also using a standard concept of literature, or literary art, which includes but is not limited to poetry and fiction. "Generated" means that computer programs have actually been written and run, resulting in literary output. Sometimes the output of programs is reworked before publication in various ways – particular poems are selected, or the text is smoothed over, or more extensive editorial intervention is made. But in many cases this output is presented exactly as originally produced. In this chapter, we deal with the history of literary text generation, the rise of the author/programmer within this digital literary art practice, how computer-generated literature speaks in machine voices, and how critics and scholars can better understand what is happening in this area, now and in the future.

A sustained type of computer-generated literature practice is now becoming widespread: that of the author/programmer. Individuals have worked as both authors and programmers throughout the history of computer-generated literature. The difference in the twenty-first century is that the author/programmer is becoming prevalent, and such people are making computer-generated literature the core of their work, building on their first explorations and producing more nuanced and compelling computational literature. To deal thoughtfully with their work, critics, like author/programmers, will need to have both technical and literary abilities.

There are certainly machines that precede the literature-generating computer, such as Ramon Llull's early fourteenth-century *Ars Magna*, allowing the generation of philosophical propositions with paper volvelles.[1] There are also a wealth of fictional text-generating mechanisms, such as The Engine in Jonathan Swift's *Gulliver's Travels*, George Orwell's Versificator in *Nineteen Eighty-Four*, *and* The Great Automatic Grammatizator in Roald Dahl's short story of the same title.[2] Authors including Stanisław Lem[3] and Italo Calvino[4] have also offered intriguing speculative discussions of how computer-generated literature might develop. Additionally, there are many present-day applications of text-generation technologies that are not utilized for literary purposes. There have been many companies in recent years (not only OpenAI, but earlier ones, including Narrative Science and Automated Insights) that have provided generation technologies for nonfictional and journalistic purposes. These do have some implications for literary studies and notions of authorship.[5] Because this generation is not directed at literary art, however, it is beyond our scope.

We have sought different threads of practice, and looked to distinguish those cases where computer-generated literature is the outcome of a unified process of programming and writing by author/programmers. By doing so, we mean to highlight an increasingly prominent practice that has been neglected in critical and theoretical discussion. It is also possible to slice computer-generated literature for study in other ways, as some have done. One could focus on a particular genre, such as poetry or prose; keep to a particular historical era; or sort literary systems based on the media elements they use, or based on how their fundamental algorithms work. There are also different contexts of presentation, including the format of the printed book, gallery and museum installation, and presentation online, which could be used as an organizing principle. While these are worthwhile ways to look at generated literature, we aim to fill a gap and describe the growing community of individual and distributed author/programmers. Those working in this community of practice have a variety

[1] Anthony Bonner, *The Art and Logic of Ramon Llull: A User's Guide* (Leiden: Brill, 2007); Michelle Gravelle, Anah Mustapha, and Coralee Leroux, "Volvelles," *ArchBook: Architectures of the Book*, December 1, 2012, https://drc.usask.ca/projects/archbook/volvelles.php.

[2] Mario Aquilina, "Text Generation, or Calling Literature into Question," *Electronic Book Review* produced by po, June 27, 2017, https://electronicbookreview.com/essay/text-generation-or-calling-literature-into-question/.

[3] Stanisław Lem, "Juan Rambellais et al., *A History of Bitic Literature, Volume 1*," in *Imaginary Magnitude* (San Diego: Harcourt Brace Jovanovich, 1984), 39–76.

[4] Italo Calvino, "Cybernetics and Ghosts," in *The Uses of Literature* (San Diego: Harcourt Brace & Company, 1986), 3–27.

[5] Leah Henrickson, "Natural Language Generation: Negotiating Text Production in Our Digital Humanity," *Proceedings of the Digital Humanities Congress 2018* (Sheffield: The Digital Humanities Institute, 2018). www.dhi.ac.uk/books/dhc2018/natural-language-generation/.

of approaches to generating literature, working to engage with computing more explicitly and profoundly than "prompt engineers" who are trying out companies' new systems and publishing the results.

11.2 Three Threads in Twentieth-Century Generation

Three significant threads of practice can be seen in the twentieth century: early programs that produced short-form outputs such as love letters, stanzas, and sentences, all formally similar; more elaborate story generators that were the academic projects of computer scientists; and more elaborate and longer-form literary text generators produced by poets and writers.[6]

11.3 Early Work in Short Forms

Author/programmers have been developing short-form projects for decades. One famous example of computerized text generation is seen in the collaboration of Brion Gysin and Ian Sommerville (one artist/poet, one programmer), who permuted lines of four or five words in the 1960s by computer – following earlier permutation poems that Gysin had written manually.[7] Such collaborations remain productive, although there have also been individual author/programmers working by themselves on short-form computer-generated literature, even before Gysin and Somerville collaborated.

Christopher Strachey seems to be the first to have undertaken creative text generation using a general-purpose computer. Strachey wrote a parody love letter generator (its first outputs were apparently displayed in 1953) that was both a humorous intervention and discussed in an art journal.[8] Strachey's other creative computing work includes a draughts (checkers) player and the first computer music, both programmed before his love letter generator.[9]

[6] These and many other sorts of text generation systems can be experienced via the text they produced in *Output: An Anthology of Computer-Generated Text, 1953–2023*, edited by Lillian-Yvonne Bertram and Nick Montfort and coming from the MIT Press and Counterpath in August 2024. Here, we provide a brief, synthetic discussion of three types of significant work.

[7] Christopher T. Funkhouser, *Prehistoric Digital Poetry an Archaeology of Forms* (Tuscaloosa: The University of Alabama Press, 2007); David Pocknee, "The Permutated Poems of Brion Gysin," 2019, http://davidpocknee.ricercata.org/gysin/.

[8] Christopher Strachey, "The 'Thinking' Machine," *Encounter*, October 1954; Noah Wardrip-Fruin, "Christopher Strachey: The First Digital Artist?" *Grand Text Auto*, August 1, 2005, https://grandtextauto.soe.ucsc.edu/2005/08/01/christopher-strachey-first-digital-artist/.

[9] Alexander Smith, "The Priesthood at Play: Computer Games in the 1950s," *They Create Worlds*, February 2, 2017, https://videogamehistorian.wordpress.com/2014/01/22/the-priesthood-at-play-computer-games-in-the-1950s/; Jack Copeland and Jason Long, "Restoring the First Recording of Computer Music," *The British Library*, September 13, 2016, https://blogs.bl.uk/sound-and-vision/2016/09/restoring-the-first-recording-of-computer-music.html.

Theo Lutz programmed the German *Stochastic Texts* (1959) to generate what read like propositions of second-order logic, using a lexicon drawn from Franz Kafka's *The Castle*.[10] Victor H. Yngve, who worked in machine translation, wrote a program to generate random sentences (1961) that used text from a children's book and was considered, by some, poetic.[11] While these two people are mainly identified with computing, the same cannot be said for poet and Neoavanguardia artist Nanni Balestrini, Nobel Prize winning writer J. M. Coetzee, or Fluxus artist Alison Knowles. Balestrini's first significant work was the 1961 *TAPE MARK I*, an Italian generator of centos with its output shown on video.[12] Coetzee wrote a combinatorial program with an extensive vocabulary that generated five-word lines (*c*.1962–5).[13] Knowles, after learning some FORTRAN from composer James Tenney, worked in collaboration with him to program *The House of Dust* (1967), which endlessly produces quatrains describing houses around the world.[14]

By the late 1960s the programming language BASIC (Beginners All-purpose Symbolic Code) had been developed at Dartmouth College.[15] It was used for many diversions, including short-form literary text generation. In one of very many examples, a program of unknown authorship simply called POETRY was included in a book published in 1975.[16] BASIC was implemented on microcomputers and became the lingua franca of personal computing.[17] The microcomputer era saw the publication of a book of poems generated by a BASIC computer program for the TRS-80 Color Computer (1981), with the source code for the program included in the back.[18] One BASIC "folk program" for Commodore computers

[10] Theo Lutz, "Stochastische Texte," *Augenblick* 4.1 (1959): 3–9, www.stuttgarter-schule.de/lutz_schule_en.htm.

[11] Margaret Masterman, "The Use of Computers to Make Semantic Toy Models of Language," in *Astronauts of Inner-Space: An International Collection of Avant-Garde Activity* (San Francisco: Stolen Paper Review, 1966), 36–37.

[12] "TAPE MARK 1, Nanni Balestrini: Research and Historical Reconstruction," Museo dell'Informatica Funzionante, June 30, 2017, https://museo.freaknet.org/en/tape-mark-1-nanni-balestrini-ricerca-ricostruzione-storica/.

[13] Rebecca Roach, "The Computer Poetry of J. M. Coetzee's Early Programming Career," *Ransom Center Magazine*, June 28, 2017, https://sites.utexas.edu/ransomcentermagazine/2017/06/28/the-computer-poetry-of-j-m-coetzees-early-programming-career/.

[14] Funkhouser, *Prehistoric Digital Poetry*.

[15] John G. Kemeny, *Man and the Computer* (New York: Scribner, 1972).

[16] "POETRY," in *101 BASIC Computer Programs* (Maynard: Digital Equipment Corporation, 1975), 169–71.

[17] Nick Montfort, Patsy Baudoin, John Bell, et al., *10 PRINT CHR$(205.5 RND(1)); : GOTO 10* (Cambridge, MA: MIT Press, 2013).

[18] Ron Clark, *My Buttons Are Blue, and Other Love Poems from the Digital Heart of an Electronic Computer* (Woodsboro: ARCsoft Publishers, 1982).

generates what can be read as concrete poetry and is only a single line long.[19] People with established poetry practices began to write BASIC programs, too, with prominent early examples written for the Apple II in the 1980s by bpNichol[20] and Geof Huth.[21] Both produce animated visual output, and are a bridge to the more extensive projects discussed later in the chapter.

11.4 Story Generation Research

Sometimes the focus of generation projects is on computing, with researchers seeking to demonstrate aspects of computing or the mind. James Meehan's TALE-SPIN, Scott Turner's MINSTREL, and Michael Lebowitz's UNIVERSE are examples of systems meant to model something general about thinking, writing, or narrative. TALE-SPIN was the first major academic project in story generation, an interactive storytelling program developed by Meehan for his 1976 PhD dissertation.[22] Noah Wardrip-Fruin has described it as having a fascinating and complex underlying model which, unfortunately from an aesthetic standpoint, is not revealed in the simple surface texts that are generated.[23] Turner's MINSTREL and Lebowitz's UNIVERSE were both developed during the 1980s. MINSTREL[24] focused on the simulation of goal-directed human authorial behaviors, while UNIVERSE[25] was designed to generate continuing serials, or never-ending stories within a universe. At the very end of the century, another research system, MEXICA, developed by Rafael Pérez y Pérez and described in his 1999 dissertation, was specifically designed to automate a model of the human creative writing process. MEXICA was developed as a research system, to inquire into the nature of creativity, but in 2017 the curated output of an updated system was

[19] Montfort et al., *10 PRINT*.
[20] bpNichol, *First Screening: Computer Poems*, 1984. Republished in *Vispo,* ed. Jim Andrews, Geof Huth, Lionel Kearns, Marko Niemo, and Dan Waber, March 2007, http://vispo.com/bp/.
[21] Geof Huth, "Endemic Battle Collage, 1986-1987," in *Electronic Literature Collection, Volume 2*, ed. Laura Borràs, Talan Memmott, Rita Raley, and Brian Stefans, Cambridge, MA: Electronic Literature Organization, February 2011, http://collection.eliterature.org/2/works/huth_endemic_battle_collage.html.
[22] Noah Wardrip-Fruin, "The Story of Meehan's TaleSpin," *Grand Text Auto*, September 13, 2006, https://grandtextauto.soe.ucsc.edu/2006/09/13/the-story-of-meehans-tale-spin/.
[23] Noah Wardrip-Fruin, *Expressive Processing: Digital Fictions, Computer Games, and Software Studies* (Cambridge, MA: MIT Press, 2009).
[24] Noah Wardrip-Fruin, "Turner's Minstrel Part 1," *Grand Text Auto*, March 1, 2006, https://grandtextauto.soe.ucsc.edu/2006/03/01/turners-minstrel-part-1/.
[25] Noah Wardrip-Fruin, "Lebowitz's Universe Part 1," *Grand Text Auto*, March 4, 2006, https://grandtextauto.soe.ucsc.edu/2006/03/04/lebowitzs-universe-part-1/.

published as *Mexica: 20 Years–20 Stories* as part of the Using Electricity series. Pérez y Pérez's involvement with the MEXICA system changed from solely that of the programmer to that of the literary author/programmer, whose name is now on the spine of a book categorized on the back cover as "Fiction/Artificial Intelligence."

11.5 Extensive Projects by Poets and Writers

The first published book of computer-generated writing seems to be *La Machine à écrire* (1964),[26] which contains a selection of unedited outputs from a text-generation project created by engineer and linguist Jean Baudot. He used a simple grammar and a constrained lexicon of 630 words to computationally generate phrases. Although this project was not necessarily created for literary purposes, the output of Baudot's program appeared in the book with contextualizing commentary by poets and writers including Gatien Lapointe and Oulipo cofounder Raymond Queneau.

Many later projects were undertaken by poets and writers. The most well-known is likely *The Policeman's Beard Is Half Constructed* (1984), a book of computational prose and poetry generated by a program called RACTER that was coauthored by writer and programmer William Chamberlain along with the help of programmer Thomas Etter.[27] Authorial credit is given to RACTER, a system that explores the grammatical structure of the English language with a dynamism propelled by random number generation.[28] RACTER was by no means a short-form text generator, and was also not a project created with the purpose of academic research in mind. It was an extensive project by an author/programmer who wished to explore the creative potential of computer-generated text. Some criticism of *Policeman's Beard* in the 1990s took the position that the project was a trick, actually done by a human author.[29] Espen Aarseth offered in response that computer-generated literature should be called "cyborg literature," giving this category of writing the tentative definition "literary texts produced by a combination of human

[26] Jean Baudot, *La Machine à Écrire: Mise En Marche Et Programmée Par Jean A. Baudot* (Montréal: Les Éditions du Jour, 1964).
[27] RACTER, *The Policeman's Beard Is Half Constructed* (New York: Warner Books/Warner Software, 1984).
[28] Bill Chamberlain, "Getting a Computer to Write About Itself," in *Digital Deli: The Comprehensive, User-Lovable Menu of Computer Lore, Culture, Lifestyles and Fancy*, ed. Steve Ditlea (New York: Workman Publishing Company, 1984), 172–173.
[29] Espen Aarseth, *Cybertext: Perspectives on Ergodic Literature* (Baltimore: Johns Hopkins University Press, 1997).

and mechanical activities."[30] The introduction to *Policeman's Beard* claims that "Once it's running, RACTER needs no input from the outside world." Aarseth found this at least misleading, as the work is not completely autonomous: Chamberlain's creative efforts are embodied in dialogue templates, the writing of source texts, and the selection of generated output for the printed book, activities Aarseth organizes into preprocessing and postprocessing, as he notes that coprocessing is also possible.[31] This, however, is not inconsistent with the claim in the introduction, which simply states that the version of the RACTER program used to produce the book doesn't take any user input after it has started running. (To muddy the waters further, there was also a commercial version of the RACTER software for home computers, not the version of the program that generated the book, which *was* interactive.) The perspective that critics and authors take on generated literature today, whether it is produced by bots that used to operate on Twitter or printed in books from poetry presses, acknowledges that there are always some "cyborg" aspects and that text cannot arise from a computer without some human involvement.

Several twentieth-century computer-generated books were produced by one author with the substantial programming done beforehand by others. An example is John Cage's *Anarchy: New York City – January 1988*, published posthumously in 2002. It consists of twenty mesostic poems generated using MESOLIST, a program written by Jim Rosenberg, and ic, an I Ching program written by Andrew Culver.[32] Having some familiarity with computing, poet Charles O. Hartman began a series of experiments with programs that were in line with his literary interests in the prosody of free and metrical verse. He developed a system to automate the process of scansion, and continued along these lines until publishing a book of poems in 1995 called *Sentences*. The poems in the book were generated in part by TRAVESTY, a system by Hugh Kenner, who is listed as a coauthor. Also used was a program Hartman devised, DIASTEXT, which automates the diastic writing strategy of poet Jackson Mac Low. Hartman's work demonstrates a personal literary exploration by way of computation that resulted in not only printed output in the form of poetry but also computer programs that engage with literary questions and forms of analysis.[33] As compelling as many of these twentieth-century projects were, they

[30] Ibid., 134. [31] Ibid., 135.
[32] Andrew Culver, "John Cage Computer Programs," Anarchic Harmony Foundation, www.anarchicharmony.org/People/Culver/CagePrograms.html.
[33] Charles O. Hartman, *The Virtual Muse: Experiments in Computer Poetry* (Middletown: Wesleyan University Press).

were often isolated experiments. In the twenty-first century, author/programmers more frequently engage in a sustained type of practice, and the emergence of communities of practice encourage and support further work in this area.

11.6 New Directions in Generation

There are now numerous author/programmers creating literary work that depends on a wide variety of computational approaches – not just the combinatorial and permutational work of Strachey, Lutz, Knowles, or Gysin, but many explorations involving, for instance, machine learning, word/image juxtapositions, performance, small-scale programs, large language models, remix, and free software culture. There is no single sort of author/programmer, as work is advancing in many new directions.

11.7 Communities of Practice

Among the several developing communities of practice, makers of Twitterbots[34] devised systems that cleverly intervene in social media feeds, producing computer-generated literature for this context. Twitter (now X) proved an amazing context for creative work in the early twenty-first century. Eleven good examples of bots, discussed and archived in an online anthology,[35] include:

- Allison Parrish's @everyword, which tweeted every word in an English lexicon from 2007 to 2014,
- Everest Pipkin's @tiny_star_field, a generator of visual poems,
- Darius Kazemi's conflation system @TwoHeadlines, the output of which included "China's Lawyer to Unveil New Evidence on Colbert Show,"
- Zach Whalen's @ROM_TXT, which finds runs of text in ROM data from videogames, and
- Ranjit Bhatnagar's @pentametron, which locates tweets that happen to be in iambic pentameter, and happen to rhyme, and pairs them into couplets.

[34] Tony Veale and Mike Cook, *Twitterbots: Making Machines That Make Meaning* (Cambridge, MA: MIT Press, 2018).

[35] "Bots," *Electronic Literature Collection, Volume 3*, ed. Stephanie Boluk, Leonardo Flores, Jacob Garbe, and Anastasia Salter (Cambridge, MA: Electronic Literature Organization), February 2016, http://collection.eliterature.org/3/collection-bots.html.

Some bots with literary and artistic aspects are nonfictional, as with @censusAmericans, which presents very short factual statements about Americans drawn from census data but "textualized" in forms such as "I haven't moved recently. I work for a private company. I was widowed."[36] There are also several fine literary bots that originate as academic research projects in computational creativity.[37] Botmakers were, of course, in touch with each other and aware of each other's work through the social network they were using. After a large-scale purge of bots from Twitter in 2018, some botmakers moved to the federated social network Mastodon.[38] Twitter became even more inhospitable for creative bots after Elon Musk's disastrous takeover of the company at the end of October 2022. With the rebranding of the company, the last creative bots were eradicated.

NaNoGenMo (National Novel Generation Month) is the main annual event in which people develop computer-generated literature. It was conceived by Kazemi in 2013, who was riffing on NaNoWriMo (National Novel Writing Month) and who included the constraint from NaNoWriMo that "novels" be at least 50,000 words in length.[39] This informal framework is welcoming and encourages offhand projects and experimentation. It also encourages people to return year after year and to develop a practice in literary generation. The technical and conventionally literary aspects of NaNoGenMo are reflected in the requirements for participation: sharing one's code and producing and sharing the textual output. Some remarkable books generated in recent Novembers include Leonard Richardson's *Alice's Adventures in the Whale*, in which dialog from *Moby-Dick* replaces that in *Alice in Wonderland*;[40] Liza Daly's *Seraphs*, a book intended to look like the undeciphered Voynich Manuscript;[41] visually innovative works, such as *i've never picked a protected flower* by Pipkin; and Nick Montfort's *Hard West Turn*, which pieces together English and Simple English Wikipedia entries related to gun violence in the United States. With the results consolidated online on GitHub, and

[36] Jia Zhang, "Introducing CensusAmericans, a Twitter Bot for America," *FiveThirtyEight*, July 24, 2015, https://fivethirtyeight.com/features/introducing-censusamericans-a-twitter-bot-for-america/.

[37] "The Best of Bot Worlds," Creative Language Systems Group, http://afflatus.ucd.ie/.

[38] Rob Dozier, "Twitter's New Developer Rules Might End One of Its Most Enjoyable Parts," *Slate Magazine*, August 8, 2018, https://slate.com/technology/2018/08/twitters-new-developer-guidelines-might-end-fun-bot-accounts.html.

[39] Josh Dzieza, "The Strange World of Computer-Generated Novels," *The Verge*, November 25, 2014, www.theverge.com/2014/11/25/7276157/nanogenmo-robot-author-novel.

[40] Leonard Richardson, "In Dialogue," November 18, 2013, www.crummy.com/software/NaNoGenMo-2013/.

[41] Liza Daly, *Seraphs: A Procedurally Generated Mysterious Codex* (San Francisco: Blurb, 2014).

the resulting texts frequently discussed by bloggers and journalists, NaNoGenMo has helped to foster new communities, critical and artistic.

Publishing computer-generated literature in print helps to raise the visibility of this form and also furthers the growth of such communities. Author/programmers often self-publish books and chapbooks, but books have also recently been put out by established presses. Named after one of the lines in Knowles's *The House of Dust,* Using Electricity is a series of computer-generated books published by Counterpath that critically engages with the long history of computer-generated literature while building a community of authors and introducing this type of work to readers through events such as group readings on the East Coast of the United States. Others have also had public gatherings to engage with interested readers and connect fellow author/programmers. Jhave Johnston, for instance, undertook a year-long project of human–computer collaboration, rising each day to revise the output of a computational poetry system he programmed. His project, titled *ReRites* and published by Anteism, includes essays by nine others and is contextualized by its own critical community. Johnson has done several readings from the project, as well as performances and recordings. Another performative project is Ross Goodwin's *1 the Road,* a reference to Jack Kerouac's book and road trip, published by Jean Boîte Éditions. The book was produced using an instrumented car that collected locative, visual, audio, and textual data. This seeded a long-short-term-memory recurrent neural network[42] that generated short texts related to particular moments. The project, supported by Google, was well documented, and Goodwin promoted it in public blog posts and offered a GitHub repository of his code.

Documentation is one of the key ways that author/programmers are building community and inspiring new projects. This may involve outlining a creative process in a blog post, creating repositories on GitHub or other code-sharing websites, publishing and distributing zines, or even developing frameworks or libraries to inspire new types of generative work and promote inclusivity.[43] There is also growing community-driven institutional support, whether through corporate sponsorship, publishing deals, or increased attention to creative coding in academic and teaching environments.

[42] For further details on LSTM (Long Short-Term Memory) RNN (Recurrent Neural Network), see www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/.

[43] Kate Compton, Ben Kybartas, and Michael Mateas, "Tracery: An Author-Focused Generative Text Tool," *Lecture Notes in Computer Science*, volume 9445, (2015): 154–161, https://doi.org/10.1007/978-3-319-27036-4_14.

11.8 Concise Programs, Extensive Output

In identifying threads of twentieth-century work, we described short-form projects, which had early origins, and distinguished these from the generation of more extensive and often book-length texts. In the twenty-first century the distinction between short-form and extensive projects does not hold in the same way. One can, for example, produce intentionally concise and simple programs to generate book-length literary output; indeed, in response to the complexity and obscurity of contemporary computing, Montfort has taken this approach and promoted it.[44]

Collaborations between Bill Kennedy and Darren Wershler involved developing conceptually simple programs to achieve compelling effects. One of their books, *Apostrophe*, is a compilation of statements that begin with "you are" that were collected by a web-crawling program, while another, *Update*, uses a simple substitution algorithm to mine personal RSS feeds and databases of names to generate updates from dead poets.[45] More recently, Montfort's *The Truelist* and Milton Läufer's *A Noise Such as a Man Might Make* are examples of books by individual author/programmers that were published with the code included.[46] The former includes all data and code used on a single printed page; the latter uses two source novels, but is generated from a program just two pages long.

An advantage of short programs is that, when the code is made available as free/open-source software, they can be easily reworked by other author/programmers into new projects. A poetry generator that has undergone very extensive "remix" is Montfort's "Taroko Gorge."[s] Invitations to write short programs from scratch can also be welcoming to author/programmers. During the 2019 NaNoGenMo, Montfort declared an additional special event or competition: Nano-NaNoGenMo, in which the generating computer programs are restricted to be no more than 256 characters in length.[47] Even with this severe limitation, some remarkable work was

[44] Nick Montfort and Natalia Fedorova, "Small-Scale Systems and Computational Creativity," in *Proceedings of the Third International Conference on Computational Creativity*, ed. Mary Lou Maher, Kristian Hammond, Alison Pease, et al. (Dublin: Association for Computational Creativity; May, 2012), 82–86.

[45] Bill Kennedy and Darren Wershler, *Apostrophe* (Chicago: ECW Press, 2006); Bill Kennedy and Darren Wershler, *Update* (Montreal: Snare, 2010).

[46] Nick Montfort, *The Truelist* (Denver: Counterpath, 2017); Milton Läufer, *A Noise Such as a Man Might Make* (Denver: Counterpath, 2018).

[47] Gregory Barber, "Text-Savvy AI Is Here to Write Fiction," *Wired*. November 22, 2019, www.wired.com/story/nanogenmo-ai-novels-gpt2/.

produced. An example is Martin O'Leary's *Dublin Walk*, generated by linking together words from *Ulysses* and *Dubliners* based on whether they are anagrams or near-anagrams.[48]

11.9 The Prevalence of Author/Programmers

Some author/programmers working today can be identified as having started their work as poets, writers, and artists. Among them are John Cayley, J. R. Carpenter, and Talan Memmott, whose digital poetry practices originate in the 1990s, with or (in Cayley's case) prior to the World Wide Web. Similarly, some author/programmers started to investigate the intersection of computation and language from the perspective of the programmer. Significant effort has been expended by computer scientists in developing natural language generation systems, but there are also literary generation projects undertaken by programmers, within and beyond the context of NaNoGenMo. Some who work in computer-generated literature are professionally software engineers, including Daly, but also have extensive background in literary computing – developing interactive fiction, in her case, as well as working in the related area of digital publishing for many years.

While the "divide" of the slash between author and programmer does have meaning, many current author/programmers did not begin on one side or the other, but have been involved with both literary authorship and programming more or less from the beginning of their practice. For these people in particular, computation is not mainly an Other to be confronted, but a medium of art. Of the people mentioned already, we will venture to claim this is the case at least with Bhatnagar, Goodwin, Läufer, Parrish, Pipkin, Whalen, and Zilles.

One result of the increasing sophistication of author/programmers is that they have been able to make literary use of cutting-edge machine learning techniques, which take a statistical approach and improve their performance as additional data is provided. Johnson and Goodwin provide two examples. Another is offered by Sofian Audry, who displayed how the results of different epochs of training unfolded in his *for the sleepers in that quiet earth* (2019), the product of a deep recurrent neural network using only *Wuthering Heights* as data. Author/programmers have also made extensive use of word embeddings (in which words are represented as vectors), with

[48] Martin O'Leary, "Dublin Walk · Issue #102 · NaNoGenMo/2019," *GitHub*, November 2019, https://github.com/NaNoGenMo/2019/issues/102.

Word2vec models being of recent interest. Transformer-based approaches, in combination with pre-training on vast amounts of text, have been the latest major advance. The main large language models used in creative text generation are from OpenAI and include GPT-2, GPT-3, and ChatGPT (backed by GPT-4), which is trained by people to "align" with corporate purposes. Free and open models have been developed in addition to these proprietary ones. One of the early open access large language modes was GPT-NeoX 20B, a formidable model that is free by design and can be used by researchers (and artists) without restriction.

11.10 Machine Voices Wake Us

Poetry, and certainly modernist poetry, has been obsessed with "human voices" – the ones mentioned in the last line of "The Love Song of J. Alfred Prufrock." Eliot even initially thought to give *The Waste Land*, with its dialectal expressions, the title *He Do the Police in Different Voices*. In computer-generated literature, the machine "does" the voices, but they are not always imitations of human ones. To adhere to our law enforcement theme for a moment, the second, short poem in the computer system RACTER's book *The Policeman's Beard Is Half Constructed* reads:

> Awareness is like consciousness. Soul is like spirit.
> But soft is not like hard and weak is not like
> strong. A mechanic can be both soft and hard, a
> stewardess can be both weak and strong. This is
> called philosophy or a world-view.

The bland statements about similarity in the first three sentences seem, from today's perspective, like something that would be straightforwardly generated from word embedding data, or from a computerized lexical resource such as WordNet. Although they are felicitous examples of the English language, they certainly do not have dialectical nuance in them. Instead, like the rest of the poem, they wear their computer-generated nature proudly on their sleeve. This is an example of a machine voice, not a human voice. It showcases how computers think and speak, at least inasmuch as the creators of RACTER imagined this. And how is that? Computers understand similarities (near-synonymy) and oppositions (antonymy), but they also understand that people are complex entities that can embody supposedly opposite characteristics. They use simple declarative sentences. The computer circa 1984 also seems to have some perspective on gender, naming a stereotypically male profession (mechanic) and using a female-gendered term for flight attendant. This, we might say, is a hint of what is called philosophy or a world-view – and, in particular, a machinic one.

An example that is similarly formulaic but will likely seem even less human, as its grammar and lexicon are more constrained, is provided by Darby Larson's 9,450-word story "Pigs" (2011),[49] which begins:

> The pig pigged with the pigs. The pigs pigged in pens. The pigs pigged on pigs. The pigs pigged in pens. The pigs pigged on pigs. The pig pigged over pigs. The pig pigged with the pigs. The pigs pigged in pens. The pigs pigged with the pigs.

The story is generated by permuting lists of starting nouns, verbs, and prepositions with final nouns. There are only four or five options in each of these lists, and the resulting work exhausts all possible permutations of sentence structure, with added wrappers to shuffle the presentation of the generated sentences. While authors have experimented with these kinds of permutations without programming, the affordances of computation allow authors to lay bare the workings of combinatorial writing in more accurate, exhaustive, and ambitious ways. Larson's "Pigs" is one extreme example, showing that with a simple grammar and limited lexicon, one can achieve a piece of writing that is extensive, diverse, and even at times entertaining.

Machine voices do not always ring out in such formulaic ways, however. In the first part of Parrish's *Articulations*, lines of verse harvested from Project Gutenberg are automatically linked together based on phonetic similarity. The result resonates with the concern for similarity of sound in traditional poetry, but creates connections that neglect sense and differ from those humans have made:

> And like a dream sits like a dream: sits like a queen, shine like a queen.
> When like a flash like a shell, fled like a shadow; like a shadow still.
>
> Lies like a shadow still, aye, like a flash o light, shall I like a fool, quoth he, You shine like a lily like a mute shall I still languish, – and still, I like Alaska.[50]

To generate *Articulations*, Parrish developed a novel procedure to computationally represent phonetic similarity,[51] using that to creatively engage with language in surprising, machinic ways. Through this representation,

[49] Darby Larson, "Darby Larson: Pigs," *Calamari Press*, July 9, 2011, www.calamaripress.com/SF/X/070911_Larson.htm; Blake Butler, "If You Build the Code, Your Computer Will Write the Novel," *Vice*, September 11, 2013, www.vice.com/en_us/article/nnqwvd/if-you-build-the-code-your-computer-will-write-the-novel.

[50] Allison Parrish, *Articulations* (Denver: Counterpath, 2018), 45.

[51] Allison Parrish, "Poetic Sound Similarity Vectors Using Phonetic Features," *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (2017).

one can find the precise word that phonetically lies between two other words; for example, the phonetic halfway point between "kitten" and "puppy" is "committee."[52] This specific relationship between computation and language opens up new ways of poetic thinking, and Parrish's book is an ambulatory textualization, or articulation, of this new thought.

More semantically concerned – and probing more explicitly into machine cognition or at least the way computers view and organize data – is the book *Machine, Unlearning.* Li Zilles's generation process for this work involved using machine learning techniques to develop questions, rather than the usual declarations:

> Could LITERATURE be algorithmic in the way that psychology can be algorithmic?
> If someone teaches LITERATURE, do they teach sociology?
> What separates a leader in LITERATURE from a leader in economics?
> Can someone study LITERATURE like they study psychology?[53]

Practical text generation for informative purposes, as with many research projects in text generation, offers humanlike texts. Along these lines, computer generation systems have been used for literary hoaxes in which their output was presented as human-written.[54] Acknowledging these humanlike uses of computing with regard to literary generation, we find that computer text generation has a special applicability in the area of machine voices. There are some famous examples of human-authored text that read as computational, such as Samuel Beckett's *Watt*, written before general-purpose computing. And there are some less famous and more recent examples, such as *Cigarette Boy: A Mock Machine Mock-Epic Presented as a Proposal to The Mackert Corporation.*[55] The truly machinic investigation of machine voices is a unique contribution, however, extending more deeply and broadly into questions of computer cognition.

To see the ways in which contemporary computer-generated literature goes beyond these human-authored texts, consider another of Zilles's computer-generated books, *The Seeker*,[56] developed for NaNoGenMo

[52] Strange Loop, "'Experimental Creative Writing with the Vectorized Word' by Allison Parrish," YouTube, September 2017, https://youtu.be/L3D0JEA1Jdc.
[53] Li Zilles, *Machine, Unlearning* (Denver: Counterpath, 2018), 65.
[54] Erica T. Carter, Jim Carpenter, and Stephen McLaughlin, *ISSUE 1: Fall 2008* (forgodot.com, 2008); Jim Carpenter, "Erica T. Carter: The Collected Works," *Public Override Void*, April 17-June 10, 2004, Slought Foundation, Philadelphia, https://slought.org/resources/public_override_void.
[55] Darick Chamberlin, *Cigarette Boy: A Mock Machine Mock-Epic Presented As a Proposal to the Mackert Corporation* (Seattle: Rogue Drogue, 1991).
[56] Li Zilles, *The Seeker* ([n.p.]: thricedotted, 2014).

2014 under the nom de plume thricedotted. Each page of *The Seeker* has the appearance of a visual poem, with formats that recur in varied ways throughout the book. In it, an algorithm seems to be trying to understand humanity by reading the instructional website WikiHow (which is actually used as input by the generating system) and dreaming. That the text not only presents a machine voice, but also seems to offer a glimpse into a machine mind, is a crucial aspect for mathematician Marcus du Sautoy, who writes, with reference to *The Seeker* in particular: "This may in fact be the ultimate goal of any algorithmically generated literature: to allow us to understand an emerging consciousness (if it ever does emerge) and how it differs from our own."[57]

11.11 Critically Reading Computer-Generated Literature

> No one can read an original Blake text, or a facsimile text, and not be struck by the following fact: that such a work has set in motion two large signifying codes, the linguistic code (which we tend to privilege when we study language-based arts like knowledge and poetry) and the bibliographic code (which interpreters, until recently, have largely ignored).[58]

Jerome McGann's observation can be expanded: William Blake was an artist as well as a poet. Engagement with visual art and art history, not just the form of the book, is of course important. Similarly, we argue that understanding the work of author/programmers requires an understanding of both literary art and computation. To understand computer-generated literature, understanding the formal and material nature of the computer as a symbol-manipulating machine – knowing something about how to program it – is essential. Otherwise, as would be the case with Blake, we can at best understand a sequence of words, not the work.

Montfort has developed a preliminary typology of computational writing that distinguishes systems based on "whether they sample or enumerate; whether they use a static or dynamic supply of source texts (or textons); their level of complexity; and their use of text only or multiple media." Each of these four axes are orthogonal, and thus each of these significant aspects is independent. This helps to show why characterizing a project with a single term (e.g., "random" or "multimedia") is inadequate. In

[57] Marcus du Sautoy, *The Creativity Code* (Cambridge, MA: Belknap Press of Harvard University Press, 2019), 265.
[58] Jerome McGann, *The Textual Condition* (Princeton: Princeton University Press, 1991).

explaining this typology, in the context of a book on poetics, a small amount of code (five lines of Python, two lines of BASIC) makes the discussion concrete. The two very short programs explain enumerating all combinations of text and sampling from a uniform and nonuniform distribution.[59]

Critical readers of computer-generated literature will need to come to terms with more code than this, because code, rather than language, is an essential medium for computer-generated literature. The author/programmer (or collaborative team) develops a text-generating system by writing code that runs on a particular computational platform, for instance the Manchester Mark I computer (in Strachey's case), FORTRAN IV (in Alison Knowles's case), the Apple II and Applesoft BASIC (in bpNichol and Huth's case) or Python (used by many currently). This particular code, with specific variable and function names, and particular implementations of different algorithms, has a form and function. That is, it works in a particular way, possibly taking data as input. Above the level of form and function, the system presents some sort of interface, graphical or command-line. Finally, there is a level of reception and operation, where a person considers how to interact and responds to the result.[60] A complete understanding of a work of computer-generated literature will include not only an analysis of the code that implements the system, but also an awareness of the platform's affordances. This is the major idea behind Platform Studies, initiated by Montfort and Ian Bogost.[61]

Another important approach is Critical Code Studies (CCS), described by Mark C. Marino in 2006 and developed in the online Critical Code Studies Working Groups run since 2010 by Marino and Jeremy Douglass. Those working in this area argue that to understand any culturally significant computer systems, poetic or otherwise, an understanding of programming is important and must be joined to more conventional types of critical reading ability. Marino's manifesto, for instance, states:

> CCS will require the artful combination of knowledge of programming languages and knowledge of interpretive approaches. These analytic projects will require programmers to help open up the contents and workings of

[59] Nick Montfort, "Conceptual Computing and Digital Writing," *Postscript: Writing After Conceptual Art*, ed. Andrea Andersson (Toronto: University of Toronto Press, 2018), 197–210.

[60] Nick Montfort, "Combat in Context," *Game Studies* 6.1 (December 2006), http://gamestudies.org/0601/articles/montfort.

[61] Ian Bogost and Nick Montfort, "Platform Studies: Frequently Answered Questions," in *Proceedings of the Digital Arts and Culture Conference*, Irvine, CA, December 2009; Nick Montfort and Ian Bogost, *Racing the Beam: the Atari Video Computer System* (Cambridge, MA: MIT Press, 2009).

> programs, acting as theorists along with other scholars, as they reflect on the relationships between the code itself, the coding architecture, the functioning of the code, and specific programming choices or expressions, to that which it acts upon, outputs, processes, and represents.[62]

Marino now holds that, rather than requiring that programmers and critics join together, critics with knowledge of programming will help drive CCS forward. Marino is such a critic himself, and writes in his recent book on CCS, which includes large amounts of code, "it is my hope that the book will be intriguing enough to nonprogrammers to draw them deeper into the study of computers and programming languages, for programming is one of the key literacies of our age."[63]

Platform Studies and Critical Code Studies are not specific to computer-generated literature, but they provide important methodologies for critics of this sort of work. By joining technical knowledge to interpretive ability, those reading this sort of literature will be able to bring it fully into the discourse on human writings and other sorts of creative computing.

[62] Mark C. Marino, "Critical Code Studies," *Electronic Book Review*, December 4, 2006, http://electronicbookreview.com/essay/critical-code-studies/.
[63] Mark C. Marino, *Critical Code Studies* (Cambridge, MA: MIT Press, 2020).