AI: Winter of Our Discontent

Richard P. Gabriel

November 18, 2024

(SWITCH to slide 2)

Every time you hear or read a story, you should ask how reliable the teller is. I am going to tell you how AI unfolded as I observed it. I am first and foremost an AI person. In this talk I'll weave in my story as it relates to AI. Figure out whether you trust me as teller.

The slides contain more material than I will talk about. Sometimes you'll see text in red; that's the only text you might want to look at during the talk. You have the slides, so read them at your leisure. After this talk, I'll give you the transcript.

(SWITCH to slide 3)

I was born in Haverhill but grew up on a farm in Merrimac. Pentucket at that time was one of the worst high schools in the country, rivaling those in Mississippi.

The next part is a long story; it started in Autumn 1966 and continued through Spring 1967. What I did, the consequences, and how I reacted and eventually recovered set the stage for the rest of my career and personality. It forms the basis of how you should judge my version of AI history.

In Senior year we had to write a term paper—jointly supervised and graded by the Contemporary Civilization teacher and the English teacher. I chose for my topic the megalopolis, the giant city.

Through the fall I worked on it. In November I was accepted at Northeastern as part of its early acceptance program—back then they took everyone. I had also applied to Harvard and MIT.

In early December I got what I thought was great news. A Harvard committee to assess students' scholarship potential sent someone to visit me and my family.

Feeling my oats, I decided to indulge my urge to write and planned to write a preface to the term paper to make concrete the abstract things my paper said about the megalopolis. I asked the teachers for permission and they said "no," but I could write a one sentence dedication.

So, I wrote a 3,000-word, one-sentence dedication, addressed to a typical resident of the megalopolis: battered by shallow culture, poor jobs, fast pace, and low reward, someone whose life is barely worth living and whose creative and living juices are throttled by the routine of surviving in a place crowded to near-death—he is a man unfortunately trapped by the horrors of *contemporary civilization* and made less a man, less a person by his circumstances.

I turned it in early.

The English teacher gave me an A+++++++: "A" followed by seven pluses. The Contemporary Civilization teacher gave me an F, averaging to C−.

Joe Sherry didn't stop there.

He wrote a letter to the Harvard admissions committee, writing that I had neither ethics nor morals, that he and other teachers suspected I had cheated in school—that my destructive attitude would damage my likelihood of success at Harvard and hurt other students.

Harvard did not accept me, and neither did MIT.

I went on strike and failed each course that semester. This rebellious streak thereafter defined my personality and approach to life—I became a cynical and caustic critic, and whatever was expected, I did the opposite. As Feyerabend said: "Anything Goes."

(SWITCH to slide 4)

We got better reviews.

(SWITCH to slide 5)

By the time I graduated from Northeastern I was a so-so mathematician, an OK Fortran and assembly language programmer, and I had no clear idea about the rest of my life except that I wanted to try for grad school. I applied to the usual places for someone who thinks he has half a chance—MIT, Harvard, Berkeley, Wisconsin, and Minnesota. I wrote away for the Oxford application. All of these were in the top 20 in mathematics back then, and predictably, I was rejected by MIT, Harvard, and Berkeley.

I studied the application for Oxford for a day or two, but I couldn't figure out how to fill it out. Effective filter, I guess.

Wisconsin and Minnesota took me and offered teaching fellowships, but I had a girlfriend for the first time, and she was going to school in Boston, so I had to figure a way to stick around.

(PAUSE)

Harriet Fell joined Northeastern from MIT; she asked what interested me and my inept answer prompted her to say *ah, you're into artificial intelligence; you should talk to my friend Pat Winston who directs the MIT AI Lab*, and I did and he let me sit in an office at the Lab for a year and when the first AI Winter hit, he said *you should talk to Dave Waltz who just graduated and has taken a job at the University of Illinois and maybe you can go with him*, and I did talk to him and Dave said *let's go to Illinois and build an AI lab.* And that's what we did—or tried to do.

(SWITCH to slide 6)

The well-known "Lighthill Report" triggered the first major AI Winter in the early 1970s. Though a British report, it had an effect on DARPA and then on me. Lighthill wrote "In no part of the field have the discoveries made so far produced the major impact that was then promised."

What is an AI Winter? Funding for AI research and development dries up, though usually not completely. AI companies—if any there be—do worse or go out of business. News outlets, social media, and the public generally are down on AI. University research funding drops but doesn't go away: funders tend to be fans of particular researchers, like McCarthy and Minsky.

New projects arise and new people join, and eventually new "breakthroughs" restore AI's luster: A New Spring.

(SWITCH to slide 7)

Lighthill blamed AI researchers for failing to address combinatorial explosion—small problems might be solved but real-world-size ones wouldn't be. Back in 1972, PDP-10s were the computer of choice for AI labs. A *large* configuration had a 9 megahertz clock and 256 kilowords of 36-bit word-based memory, which translates to one megabyte.

Why "computer of choice"? Most AI researchers used Lisp, and a PDP-10 address was 18 bits, which meant a Lisp CONS cell could be stored in a single word. Moreover, there were machine instructions to manipulate these half-words, so in a sense, a PDP-10 was a Lisp machine—by design.

My computer at home today is ten thousand times faster and almost two hundred thousand times bigger... and not shared with others.

Naturally, we all knew back then we needed way bigger and much more powerful computers for "real AI." Likewise, we knew that only "toy" problems were accessible.

(SWITCH to slide 8)

Lighthill broke AI into three categories. Category A was "Advanced Automation," which addresses tasks where machines could replace people; his examples included OCR and pattern recognition.

(SWITCH to slide 9)

Category C concerns using computers to help come up with theories to interpret neurobiological data about specific areas of the central nervous system, using computer-based models of neural nets to test out particular hypotheses.

(SWITCH to slide 10)

Category B is Building Robots, that is, building devices that mimic a certain range of human functions. As of 2024, the three categories have blended, in large part because computer power has expanded the size and range of problems that can be addressed.

(SWITCH to slide 11)

Two other important critiques of strong AI came out, one in 1972 and the other in 1980, but I won't talk in detail about them. They spurred quite a kerfuffle.

Dreyfus argued that human intelligence and expertise depend on unconscious processes rather than conscious symbolic manipulation, and that these unconscious skills can never be fully captured in formal rules.

Searle argued that if understanding or intentionality cannot be ascribed to a machine, then it is not thinking; it does not possess a mind in any ordinary sense of the word.

(SWITCH to slide 12)

Where did this criticism come from? Why such a strong impact?

AI might as well have a "kick me" sign on its back. Unlike many "problems" computer scientists and programmers address, AI has an easy-to-understand goal, and the nature of success is vague—as vague as anything people try to do: as vague as success in painting, fiction writing, sculpture, music composition, and poetry.

Write a program that does things that usually only a person can do. Play a game, read a handwritten note, prove a theorem, write a program, transcribe a lecture, compose a sonnet, translate Madame Bovary to Pit-jant-jat-jara, drive a car through the Mojave from Barstow to Primm, tell cats and croissants apart.

What game, and how well can the computer play? How long a note? How hard a theorem? Reminds me of Samuel Johnson's remark, which I'll revise: "Sir, a program writing a sonnet is like a dog walking on his hind legs. It is not done well; but you are surprised to find it done at all."

Back to the beginning.

(SWITCH to slide 13)

AI started in Summer 1956 at Dartmouth College, at a two-month study proposed by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. Allen Newell and Herb Simon would make an appearance. The group was optimistic about what it could accomplish and how quickly: "problems . . . reserved for humans" and "improve themselves," and a "significant advance"—over two months in Summer.

(SWITCH to slide 14)

In 1955, there were roughly no programming languages higher level than machine language and some simple, so-called "autocode" languages, which were pretty much assembly languages.

Even though the machines were computationally tiny, the main problem was seen as an "inability to write programs" that could exploit them. Human thought consisted of manipulating words using reasoning and rules of conjecture.

(SWITCH to slide 15)

We can see the origins of several AI threads in vogue today: neural nets and using randomness including Monte-Carlo methods. They were considering self-improvement, creative thinking, educated guesses, and hunches.

(SWITCH to slide 16)

Take a look at some of the words threading through the proposal.

> *imaginative, invention, discovery, uncertainty, failure, slightly wrong, unreasonable, unexpected, rough guess, originality, self-reference, learning*

Even today, many would have trouble reckoning how to approach some of these.

Newell and Simon set the stage for years of research: because we don't even know what questions to ask, we shouldn't go for theories first but instead build actual programs to understand and study.

(SWITCH to slide 17)

Some of you are too young to appreciate or even know how primitive computer science was in the early 1970s. Don Knuth published the first three volumes of "The Art of Computer Programming" in 1968, 1969, and 1973. There were few if any undergraduate degree programs in Computer Science.

(SWITCH to slide 18)

Given the vague goal statements McCarthy and his crew put together, progress seemed pretty good. Glance at this transcript of Joseph Weizenbaum's Eliza program in 1967, whose most famous persona was a Rogerian psychologist.

(SWITCH to slide 19)

Now look at Ken Colby's Parry program chatting with Eliza. Parry had a much more sophisticated language understanding model; Parry was designed to mimic a paranoid schizophrenic. In lots of ways, Parry was a serious attempt at AI while Eliza wasn't. Take a look at this exchange and see how Parry is the better program.

A group of experienced psychiatrists analyzed real patients and computers running Parry. When asked which were human and which were computer programs, they were correct only 51% of the time—a figure consistent with random guessing.

(SWITCH to slide 20)

Terry Winograd at MIT made the most notable advance in the early 1970s with SHRDLU, a program that used a simulated blocks world as a platform for a person to interact with a "robot" in English. This is a transcript. The person would see a display of the blocks and the actions of the robot. SHRDLU seems to "understand" what's going on and seems able to reason about the blocks world.

Winograd produced two programming systems to implement SHRDLU: Microplanner is a backtracking theorem prover, used for reasoning and planning; Programmar is for natural language understanding.

(SWITCH to slide 21)

As remarked by Newell and Simon, most of these achievements are based on guessing how to tackle the underlying "problem." Many researchers at the time lamented the lack of computer power, which forced them to use "toy" examples, and so the idea that perhaps their guesses were off base wasn't top of mind.

We—because I was in the AI crowd—were doing our best to dope out how to approach programming such ill-defined behavior.

On top of that, the notion of mimicking human mental capabilities is an easy one to grasp—more so than, for example, appreciating the difficulties of tackling NP-complete problems, such as satisfiability—and judging success seemed to be a matter of "knowing it when I see it." And seeing just a little of it made it easy to believe there was more lurking just out of sight.

CEOs and the military liked the idea of cheaper, more effective, and more compliant, nonhuman actors.

Not surprising that there was an AI Winter.

(SWITCH to slide 22)

Trying to start an AI Lab at the University of Illinois was not easy. To establish an AI degree program required passing a bill in the Illinois Legislature. For me this meant that getting a doctorate there was going to be tricky. I was in the Math department, and their plan was dramatic: Take and pass all the tests required for a Math PhD. Do research and write a dissertation in AI that satisfied both Dave Waltz and the Math department; and because they were unsure about both AI and Dave, the dissertation had to satisfy two other folks: Marvin Minsky and Seymour Papert at the MIT AI Lab.

Urk.

Dave Waltz was a favorite son in the EE department, and they had a plan: I'd take but fail all the required EE tests, and using a quirk in protocol, Dave and the EE Chair would declare that I had passed; then I'd do AI research and a dissertation that satisfied Dave Waltz. But I was still too starry-eyed to overlook the fraud.

I decided to move on. I applied to MIT (again) and Stanford—the two best AI Labs. Dave Waltz called in a favor with someone at SAIL—the Stanford AI Lab—and I was whiteballed into the CS Department. A "whiteball" is the opposite of a "blackball." I was put on the MIT waiting list. Don Knuth called to invite me to join him at Stanford. Who could say "no."

I talked McCarthy into my joining the Stanford AI Lab in exchange for TAing his Lisp course. He was the worst teacher you can imagine and just as bad at programming Lisp. I decided that despite McCarthy being at SAIL, they had a subpar Lisp, so I ported MIT's MacLisp to their PDP-10 and became the Lab's Lisp wizard. I wrote lots of libraries and two Lisp programming environments. For my wife's master's thesis I wrote a gait diagnosis program using expert system ideas.

I didn't know much about computer science, so another student in my incoming class dumped a trunkload of papers and books at my house and said "read these and you'll know computer science." None of the

18 people admitted to the CS department that year had an undergraduate computer science degree. I took four CS courses and the rest were “Reading and Research” and “Advanced Volleyball.” (Look up “Gabriel” in the Hacker’s Dictionary.)

Terry Winograd became my adviser and I joined an Automatic Programming project to do their natural language generation stuff. The night before my oral exam, Terry called up and told me he planned to vote “no.” Not great news. But another student in my incoming class had helped me prepare my talk to an absurd level (something like 80 practice runs), so I passed without any grilling by the committee. He was also the bass player in my rock ’n’ roll band.

My thesis system was called “Yakety Hacks,” but I’ve never revealed that before today. By the way, I designed and implemented a programming language to write it in along with a soft-description language.

I did a lot of Lisp hacking and became known for it.

(SWITCH to slide 23)

After graduating, McCarthy read my dissertation; despite declaring “it’s not very good,” he hired me to work on his revived Advice Taker project, where the basis of research was this statement, “in order for a program to be capable of learning something it must first be capable of being told it.”

The approach was to represent propositions in a formal, mathematical language; and the interesting “toy” problems, as it were, require non-monotonic reasoning, the fact that the mere addition of new premises can invalidate a previously reached conclusion—something that cannot happen in purely deductive reasoning. Something like this:

All Birds can fly.
Canaries are birds.
Penguins are birds.
Slick is a bird.
Can Slick fly? Yes.
Penguins cannot fly.
Slick is a penguin.
Can Slick fly? No.

McCarthy wanted us to use circumscription, his invention. We used a Neo-Fregean approach.

Around then the MacLisp community was split and drifting apart, and DARPA wanted to fund only one Lisp effort for their AI research, so I gathered the community, forged—or forced—compromises, and Common Lisp was born.

After Common Lisp was designed, I started a Lisp company to implement it on stock hardware. It was a “private label” or “OEM” company. We provided Common Lisp to every workstation and mainframe company except Tektronix, and every AI company used our Lisp.

In 1988, the second major AI Winter hit.

(SWITCH to slide 24)

In the US, work on expert systems, simple planning, and logic-based reasoning overflowed into the business world, especially when AI researchers suddenly realized a lot of money could be made. The success of MYCIN spawned a raft of companies, mostly in California.

The Japanese government launched a ten-year project to create a machine that could perform somewhere between 100 million and 1 billion “logical inferences per second,” a measurement they referred to as “LIPS.”

MYCIN used an inference engine and rules written like this one, which can be read thus:

“if the gram stain of the organism is gramnegative, and the morphology of the organism is rod shaped, and the aerobicity of the organism is anaerobic, then there is suggestive evidence (at the .6 level) that the identity of the organism is bacteroides (back-ter-óy-deez)”

(SWITCH to slide 25)

Here is a rule from my Gait Diagnosis program.

Two things to notice: the rules go from observations to diagnoses (usually) and the force of the inference is a number. Critics of expert systems back then remarked that expert systems required you to program with floating point numbers (certainty factors)—adjusting them until you got the right answers. Perhaps someone should tell that to NVIDIA.

A certainty factor is a measure of how certain the system is of a statement; they range from -1 (certainly false) to 1 (certainly true). There is a calculus of combining them.

(SWITCH to slide 26) **(Bucks Cap)**

When I started my Common Lisp company in 1984, like the AI company founders, I needed to convince venture capitalists to give me money—a lot of money. To get a lot of money without giving up most of your company, you needed to promise to make the venture capitalists a lot of money, redeemed at IPO. I recall working with my primary co-founder to come up with the most outrageous predictions of market sizes that didn't revolt us as scientists. You might say that we "boasted of a bright future."

Naturally we were confident in our ability to create a good Lisp implementation on what was called "stock hardware," but we depended on the hardware companies to make capable hardware and the AI companies to lure in customers.

And therein lies the rub.

(SWITCH to slide 27)

AI systems are large and complex; many were developed on Lisp machines, such as the Symbolics 3600, the LMI CADR, and the Xerox Dorado. These machines were designed and built by Lisp-knowledgeable people, so the special performance needs of Lisp were handled by hardware. These machines were expensive. Could Sun Workstations, for example, keep up?

(SWITCH to slide 28)

Dave Moon—arguably the best programmer alive in the 1980s—remarked that AI represented "complex, ambitious applications that go beyond what has been done before." And what expectations did the AI companies hint at? Many potential customers had cadres of aging, knowledgeable experts at the hearts of their companies, and these companies heard that expert systems could replace those experts as they retired—or were fired. The expectation was human-level performance for specialized, cognitive tasks.

(SWITCH to slide 29)

The Lisp machines were indeed powerful and easy to program with—for Lisp people, and, I found out, for those with expansive minds. My company was full of mathematicians and musicians, and even those with no Lisp background learned quickly how to "hack Lisp real good," as Rod Brooks once said. This was the early to mid 1980s.

(SWITCH to slide 30)

Hype, we know now, derives in part from an "echo chamber," in which the same kinds of statements are repeated over and over. AI fundamentally draws people to overstate goals and performance, but along with that, many AI companies were led or directed by enthusiastic researchers who loved their own cooking and liked the idea of a big house on top of a South Bay hill with a view of Stanford.

(SWITCH to slide 31)

How lame were the people running AI companies?

Around 1987 I met with Tom Kehler, the CEO of IntelliCorp, to talk about my company helping his by tailoring and tuning our Common Lisp so that their products could work better for our shared customers. He flatly refused; he said that my Lisp company had only one upward path, which was to provide AI systems in competition with Intellicorp, and that capitalism was based on non-cooperation.

By the way, this was when Lucid, my company, was subleasing our offices from Intellicorp, a deal made because of the confluence of our business interests.

(SWITCH to slide 32)

Blame.

As Supreme Court Justices and under-motivated children prove, dogs and wives are to blame. The AI companies blamed Lisp for their failures. By 1987 there were plenty of competent Lisp implementations created by a variety of good companies and backed by strong hardware companies.

(SWITCH to slide 33)

AI from the 1950s through the 1980s concerned figuring out how to write programs people had no idea how to write to accomplish the vaguest of absurdly impossible goals. As McCarthy said, the problem was "our inability to write programs."

AI was about programming.

Here is some programming stuff invented to cope with AI.

(SWITCH to slide 34)

Let's imagine. The 1950s, '60s, & '70s: you want to write a program that performs something intelligent a human could do. You aren't sure how to approach it, but you have some ideas. The computers you can access might have a megabyte of memory and perhaps 30 megabytes of disk space. They can execute a million instructions per second. Your choice of programming language is roughly these: assembly language, Fortran, Lisp, and Cobol. There are no libraries able to help; there are hardly any libraries at all.

This is how things were until the mid-1970s. Any sort of help a programming system could provide came from something you created. In short, you or your colleagues needed to create the programming languages and surrounding infrastructure yourselves.

(SWITCH to slide 35)

When your goals are correctness and performance, you invent type systems, better compilers, and modularity, and you promote formal reasoning. When your goal is figuring out how to program something that many believe cannot be programmed, you invent mechanisms of expressiveness.

Gaze upon some.

(PAUSE)

(SWITCH to slide 36)

(PAUSE)

(SWITCH to slide 37)

(PAUSE)

(SWITCH to slide 38)

This AI Winter was a disaster for computer science. To see this we must explore the idea of Programming Systems.

A programming system is a platform comprising execution hardware at the lowest level up through the interfaces to the outside world and to the programmer or user. It's a bottom-to-top-to-bottom system that supports programmers building up a system starting with an already fledged and running system that supports reflection and self-modification; it is in short a live system.

(SWITCH to slide 39)

Lisp machines, Smalltalk, and roughly every Lisp implementation exemplify programming systems. A programming system is not "just a language and an operating system."

(SWITCH to slide 40)

Work on and with programming systems produced several important ideas, methodologies, and now-common technologies.

(SWITCH to slide 41)

Some of them might strike you as unexpected, surprising even.

(PAUSE)

(SWITCH to slide 42)

My Lisp company and I noticed right away the business consequences of this AI Winter.

After I finished my poetry MFA in 1998, I saw that my research areas had been deleted by the Academy. And only in 2012 did I notice the demise of an entire way of thinking about programming and even computing.

(PAUSE)

(SWITCH to slide 43)

In the late 1980s I realized that I had neglected my passion for writing; in 1993 I attended three writers' workshops and found out that actual poets thought I might be not bad. My computer science career became only a means to support my writing, something all my employers eventually noticed.

In 1995 I started at an MFA program, graduating in 1998. I intended to resume my research career, but there was no interest in the work I wanted to do nor were there any places to publish. I became a journeyman "helper," helping Sun with some projects including transitioning to an open-source company, working with marketing, and acting as a trophy to display corporate health to the outside world.

I resented having no place to publish, so I helped design the Patterns conferences (the PLoPs), started Onward!, expanded it to include Essays, started the transition of OOPSLA to Splash, and helped design the Art, Science, and Engineering of Programming—all to create publication venues I could inhabit. Along the way I became expert in the work of Christopher Alexander.

I wrote a couple of books, ran a couple of interesting conferences, and gave a handful of interesting talks.

Around 2013 I happened upon a research project that combined my interest in writing & creativity with my knowledge of AI: InkWell. We can ignore its origin story and focus on what I made it: an exploration of how creative writers—especially poets—think about and go about revision. Poetic revision goes way beyond conveying facts, meaning, and information; it's about connotations, sound, rhythm, lyricism, mood, attitude, voice, and subtext.

(SWITCH to slide 44)

In 1948, Alan Turing described how intelligent machines could come about. The machine—software mostly—would need both *discipline* and *initiative*. Discipline comes from programming, which I take (today) to mean the symbolic AI stuff: reasoning and actual thinking.

Initiative might come from programming, but Turing hinted that one could start with an unorganized machine—a neural net, perhaps—and *teach* it discipline and initiative.

(SWITCH to slide 45)

Most AI work until the 1990s was symbolic AI, or programming. Since then machine learning has taken over—at least that's what the press, public, and Wall Street believe.

(SWITCH to slide 46)

These days we're impressed by ChatGPT and other Large Language Model-based systems. They can answer questions, write short essays and articles, and even write some programs with aplomb. We are so impressed by these abilities we are beginning to worry about our place on the intelligence spectrum. But long before these generative feats, computers using non-symbolic techniques could do some amazing things—and in many ways, these things could be called superhuman.

(SWITCH to slide 47)

We'll start with genetic algorithms. The idea is simple: simulate natural evolution with mechanisms for breeding, mutation, crossover, and selection. I use genetic algorithms for some things in InkWell.

If you're worried now about generative AI, you should have panicked in 1997 when Adrian Thompson's group did an experiment to evolve an FPGA to distinguish between two signals:

(SWITCH to slide 48)

a 1k square wave and a 10k square wave. They used a 10x10 section of a field programmable gate array, but without a clock. The FPGA was inside a Faraday cage because they had discovered that the genetic algorithm liked to design a radio in the FPGA to pick up the controlling computer's clock!

(SWITCH to slide 49)

Here's a diagram of the essential part of the evolved cell topology. Without a clock, it's an analog circuit. It worked very well.

Thompson and his colleagues tried to figure out how it worked. Best I know, they never did.

(SWITCH to slide 50)

Here are some strange things about the circuit. The training / evolution process used an actual FPGA; when a different physical part is substituted for the one trained on, the circuit doesn't work well. If you make the circuit in the diagram with CMOS, it doesn't work. If you simulate the circuit with one of the commercial simulators, it does nothing.

The function units of the red cells are not used by the circuit, but when those units are clamped—their function units set to return a constant 0—the circuit doesn't work. Ignore the upper lefthand cell in the diagram: The remaining red cells are used only as routers—that is, they are essentially wires in the circuit. That upper lefthand red cell, though, is particularly interesting: It is not directly connected to the active part of the circuit at all. Yet if its output is clamped, the overall circuit doesn't function.

(SWITCH to slide 51)

The most intriguing behavior is how the circuit gets the answer. The right answer is gotten 200ns after the trailing edge of the first high waveform, which means that whatever interesting that's happening is happening when that waveform is high. But, as far as anyone can tell, absolutely nothing is happening in the circuit at that time.

When the input goes high, the feedback loops in parts A and B are cut, and therefore those parts of the circuit revert to digital logic and become static. Part C is observed to be static.

(SWITCH to slide 52)

When the input is high, the power levels are quiescent, there is no activity in the feedback loops, there are no short pulses or glitches anywhere, there is no thermal activity, there is no RF activity nor response to RF. In short, no observation shows anything happening at all.

(SWITCH to slide 53)

Once the pulse falls, Part A starts to ring, with different frequencies depending on the length of the pulse. Parts B & C then turn that into the final output.

In 1999, in a paper in CACM, Adrian Thompson described these observations like this: "This is astonishing," "This is hard to believe," "…somehow, within 200ns of the end of the pulse, the circuit 'knows' how long it was, despite being completely inactive during it," "We still do not understand fully how it works,…" and "Artificial evolution can produce bizarre circuits that work—but do we need to understand them?"

(SWITCH to slide 54)

Genetic algorithms and neural nets—they meet in neuroevolution.

Neuroevolution evolves the topology of the neural network along with the weights associated with the connections and the rules that govern how the weights change over time.

(SWITCH to slide 55)

A tough pole balancing task is to balance two poles attached to a cart by moving the cart to keep the poles from falling—given only the cart position and pole angles.

(SWITCH to slide 56)

This clever solution works by using the recurrent connection of the single hidden node to itself to compute the derivative of the difference in pole angles, thereby figuring out whether the poles are falling away or toward each other. Without evolving structure, it would be difficult to discover such subtle and compact solutions. Starting minimally makes discovering such compact solutions more likely.

(SWITCH to slide 57)

Kenneth Stanley and his colleagues explored an evolutionary technique based purely on "search for behavioral novelty." The idea is to ignore objective-based fitness functions and use only a measure of novelty—an individual is favored when it is different from previously evolved individuals. This enforces an increase in complexity, which is a key issue in evolutionary theory.

Novelty search can solve several so-called "deceptive" tasks, such as difficult mazes. That is, evolve a neural net that is able to traverse a tough maze. Another experiment involved creating a biped that could travel the greatest possible distance from the starting location. Novelty search consistently performed better than pure objective-based searches, primarily because the novelty searches routinely discovered hip rotation.

Simulated annealing works well because it exploits a kind of novelty seeking search.

When Inkwell was instructed to write a series of short poems based on the same prompt, it eventually came up with the doozy you see here.

(SWITCH to slide 58)

Machines and computers can perform many tasks typically associated with people *much better* than people can. The human record for solving Rubik's cube is 3.13 seconds, set by Max Park in 2023. The robot record is .305 seconds, set this year by a Mitsubishi Electric robot. Both the mental aspect and actually twirling the cube. If we were to adopt the fear-mongers' pessimism, there would be no reason for a person to ever play with a Rubik's Cube—the same way that nowadays no one plays chess or Go, no one runs races, and Jeopardy! is off the air.

(SWITCH to slide 59)

On to ChatGPT and generative AI. I am not going to explain in detail how those systems work, except to point out that they are more sophisticated than many skeptics give them credit for; but I will say that when it comes to writing, their capabilities fall very short of what even modest but talented human writers can do.

Their success depends on a few things: first is astonishingly large training texts; the second is a deceptively clever representation of word meanings; and the third is a transformer-style generation mechanism based on a complex attention mechanism.

(SWITCH to slide 60)

Word2vec is a vector-based representation of word meanings, learned neural-net-style by optimizing the prediction of words in a moving window of text. Basically, the meaning of a word is reckoned from the words that it appears near in real texts. The vector has no inherent meaning—its elements aren't, for example, other words—the idea is to locate words in a sufficiently high-dimensional space so that "being near each other" implies semantic closeness, and direction implies semantic category, more or less. Dimensions I've seen range from 100 to over 12,000.

(SWITCH to slide 61)

For example, let's compute the offset vector that joins the point representing "man" to the point representing "woman"; then we'll locate a point that is that same offset from "father" and show the ten closest words to that. Not bad.

(SWITCH to slide 62)

Oops. Doesn't always work well.

(SWITCH to slide 63)

Very roughly and a bit inaccurately, a transformer takes as input a series of words or sentences, encodes the words with something like word2vec and positional information, and using a bunch of attention layers to figure out the ways various words relate to each other, spits out a series of words or sentences that are a response to the input. This response is based on predicting the next word in each sentence using meaning gleaned from the input and the sentences up to that point. The whole shebang relies on a word2vec-like representation of meaning because the next word prediction relies on shifting possible choices around based on multi-headed attention—shifting kind of like the man-woman shifts we just saw.

But note that the examples I showed used word2vec representations of size only 300, which is not great. ChatGPT uses a more elaborate and much larger multi-dimensional representation sort of like word2vec. Larger—like 12,288 instead of 300.

There are a raft of neural nets used in the transformer, and a pile of matrix multiplications. A big raft and a huge pile.

(SWITCH to slide 64)

I'm not a good enough programmer to judge how well ChatGPT programs—plus, I can program only in Lisp. But I asked ChatGPT to explain this function.

Wow: it figured it out. ChatGPT must be really, really good at staring at code and doping it out. This is better than many programmers I know can do.

But wait. . . .

(SWITCH to slide 65)

I took that function and changed the variable names to gensyms—what theorists call "alpha conversion." I started a fresh chat; ChatGPT was stumped. It seems to use a combination of surface-y reading of the source and staring at variable names to figure out the code.

On the other hand, this is just what many programmers I know actually do.

(SWITCH to slide 66)

Getting back to writing—which I'm better at—I asked ChatGPT to write a short poem responding to these words: loud guitar blues music. This task was posed to InkWell in 2015 by my now-deceased colleague and bass player, Ron Goldman (hey man!).

The three ChatGPT ones are simply not very good, though one could forgive them were they written by high school students. I believe this becomes apparent when you compare it to the one InkWell wrote.

A writer colleague said of the third one:

> *There's a lot of cliché in the poem. The material is precious. There's no sense of a concrete experience. Though the lines strive for metaphor they fall short of finding anything fresh or specific. The excessive anthropomorphizing is a problem, too.*

(SWITCH to slide 67)

Nickieben Bourbaki is a pseudonym I used in the late 1980s, most notably in a series of debates between Nickieben and me in two different tech magazines on the topic of Worse Is Better—I for, he against. Different LLMs do better or worse on this depending on whether they were trained on my website.

Quite inventive.

(SWITCH to slide 68)

Then I asked ChatGPT to revise a first draft of a poem into prose, using its own voice and making it more interesting. The original poem is mediocre but not incompetent.

That writer colleague said of it:

> *The piece starts with cliché and generalized description and moves toward anthropomorphizing the city but not in any fresh and original way. . . . There's no real attitude or tone beyond a general one. We get some specific description. I'd say "dust-covered crosses" is the strongest, but the writer can't leave well enough alone and goes too far editorializing about what those crosses mean. The detail is already significant without having to explain. . . . We are told there's a subtle humor but the piece offers no sense of it, no experience of it.*

By the way, I used the expensive version of ChatGPT. . . .

(SWITCH to slide 69)

. . . perhaps we can reason about its behavior.

(PAUSE)

(SWITCH to slide 70)

People fear LLMs will replace human writers. In fact, LLMs are decently competent casual writers—they will not replace, for example, Cormac McCarthy. I believe AI will eventually produce interesting individuals, but LLMs are not that.

One of my goals with InkWell was to see how far a program with only language knowledge could go toward thinking like a writer. LLMs share this language-centric focus. But LLMs are built to explain using only what they were trained on, what people tell them using prompts, and what a clever algorithm supplies as direction. Because of these limitations, an LLM cannot be an individual.

While we're at it, perhaps we need to rethink the Turing test: Does it show that machines are human or that humans are not so extraordinary?

(SWITCH to slide 71)

Real writing is about revision. I own and have read a dozen books teaching revision. I'd say LLMs are good at writing first drafts at the high-school term paper level.

LLMs are good at explaining facts, but they don't seem to do well with words, their noise, their rhythms, connotations, loveliness, cultural nuance, point of view, voice—things I worked on with InkWell. The poet Brenda Hillman told me: *when in doubt make it strange.*

(SWITCH to slide 72)

I had a weird thought: is the reason LLMs seem so good also why they are not very human?

A person cannot know everything, but has mechanisms for discovery and handling novel situations. LLMs don't seem to have those, quite. The only experiences they have are those that have been written down, acquired during training or in prompts.

An LLM has been trained in every point of view, so cannot have its own.

Perhaps if an LLM were more limited—more ignorant and more illiterate—it could be a better writer, or more like a person. Perhaps it could learn to forget.

(SWITCH to slide 73)

Robert Boswell—a good writer and great writing teacher—talks about an approach to fiction that nearly all creative writers know: a form of contemplation, a complex and incongruous way of thinking, working in half-knowledge.

(SWITCH to slide 74)

While revising, the writer listens to what has made it the page. The uninvited is often the most interesting: unusual landscapes and strange desires emerge when the world remains half-known.

(SWITCH to slide 75)

There can be no discovery in a world where everything is known. A crucial part of writing is remaining in the dark.

Do I dare say it? Worse might be better.

(SWITCH to slide 76)

Any chance of another AI Winter? Dunno. Depends on which AI you mean.

Speaking only of LLMs and creating texts, the current value of ChatGPT is about the same as a decently talented, relentless seventeen year-old good at using Google but a tad gullible. Irresistible business value, eh?

(SWITCH to slide 77)

But there is more to it.

Recall the late-1980s AI Winter that was a disaster for computer science— its collateral damage was the near-end of research on programming systems. The current mania over deep learning and LLMs is a disaster for symbolic AI. Without symbolic AI or something like it, artificial systems would be limited to what can be learned from large training sets. Without symbolic AI, AlphaGo wouldn't work; ChatGPT wouldn't work.

Without symbolic AI you can't create a program that tries a variety of classifiers and combines them to reach a novel conclusion. We would be unable to tell a system—Advice-Taker style— that a certain kind of unicorn is a white goat-like animal with a long straight horn with spiraling grooves, cloven hooves, and sometimes a goat's beard. Instead, we'd need to ask "where's the training set?"

Since the 1990s some researchers have been promoting "neuro-symbolic AI," which integrates neural and symbolic AI architectures to address the weaknesses of each, providing an AI system capable of reasoning, learning, and cognitive modeling. But if we are in a Symbolic-AI Winter, if funding outside deep learning and LLMs diminishes, how long will we need to wait until the next Symbolic-AI Spring?

(SWITCH to slide 78)

Thank you, good luck, and go well.

(PAUSE)

(SWITCH to slide 79)