Generative AI as Mathland and Constructionist Frontier

Logo traditions, computational fluency, and emerging technology converge to create new opportunities to amplify the potential of each learner

The recent availability of generative artificial intelligence systems, such as ChatGPT, has captured the imagination of the public and schools. Amidst the fearmongering and hype surrounding AI, its inevitability causes educators to either scream their objections into the abyss or resign themselves to being passive consumers of the technology. The work of constructionists, like Papert, Solomon, and Minsky suggests a third option; children developing sufficient computational fluency to gain agency over the technology and face the future with personal empowerment. Realizing this aspiration calls on timeless Logo traditions, respect for learners (and their teachers), and an awareness of emerging technology. This paper explores work recently undertaken with fifth thru eighth grade students and educators across the globe. The confluence of the public's fascination with new technology and calls for widespread computer science education, creates an unprecedented opportunity for constructionists to assert our pedagogical expertise, design better computing environments for learners, increase computational fluency across disciplines, and make schools better places for children.

1 INTRODUCTION

The release of ChatGPT and similar generative artificial intelligence systems will undoubtedly accelerate the development and availability of increasingly sophisticated software. Ignorance of the technology has led to an unprecedented cacophony of hype and hysteria. The only conclusion one can reach by the proliferation of LinkedIn bios in which my contacts declare themselves "*AI in Education Expert*" or the abundance of conference presentations suddenly about educational AI is that education was apparently revolutionized over the Christmas holidays of 2022. The very same edtech influencers eager to cash-in on the latest AI craze are the very people who have stood in the way of Papertian uses of computers in education for decades. Many are what Papert used to call, "idea averse." (Papert, 2000)

The rhetoric surrounding ChatGPT appears to miss the greatest arguments for constructionist learning and computer use in the style of what Seymour Papert and his colleagues advocated. In an age of rising authoritarianism, concerns of democratic erosion, accusations of fake news, and misinformation it is critical to answer Papert's question, "Does the child program the computer, or the computer program the child?" On the side of the child.

While the explosion in interest in technology may represent a positive force for those of us who have long advocated for educational computing, my fear is that "AI in education" will achieve the same fate as the pathetic efforts at "computer science for all." Although computer science for all is a noble objective, its implementation has been led by policy makers, with little actual computing expertise, resulting in vocabulary acquisition, "Hour of Code" puzzles, and denatured experiences, like Computer Science Unplugged, as if computers were unavailable.

Computing and artificial intelligence are too important and hold so much potential for progressive school improvement to be left to charlatans, textbook publishers, and tech billionaires harboring longstanding contempt for public education.

Constructionists, standing on the shoulders of the Logo community and its long-standing traditions of progressive, project-based learning experiences have an opportunity and responsibility to lead K-12 education making sense *with* and *of* artificial intelligence.

> "And as we grow up, we should stop seeing ourselves as specialists of computers in education, because that casts us in the role of a kind of service profession. Accepting the role allows that other people are the ones to decide the big goals of education, what the curriculum is, how learning happens, what's a school. And at our conferences we talk about how their decisions can be served by the computers. Well, fine, up to a point. This certainly allows revolutionary actions as long as we are at the stage of crafting Trojan horses to throw into the system. But at some point, we have a responsibility to break out of that marginal role and take on our true vocation, which is not one of service but one of leadership." (Papert, 1991)

Our allies in leading education in the age of AI, may once again be the sorts of mathematicians, scientists, computer scientists, artists, philosophers, epistemologists, and progressive educators present at the birth of Logo fifty-five years ago.

2 AN INTERVENTION

As a longtime proponent of teaching children to program computers in a Logo-like environment, I continue to push that rock up the metaphorical hill of apps, verbal inflation (Papert, 2006), and disappointment. Kids are capable of doing so much with computers but are rarely afforded the opportunity to do so. Constructing knowledge computationally would be of great benefit for what Papert called "learning learning". In Logo environments, the child does the work of the computer by thinking about how to communicate a sequence of formal instructions and then either debugging the errors in their thinking or building upon their success. This process of engaging with transitional objects, in many cases the Logo turtle, concretizes abstract concepts that are then presented to the computer formally in a reciprocal and generative "conversation." Yet, even in the "best" Logo classrooms[1], far too few students experience the epiphany associated with reaching the tipping point at which you stop working for the computer and the computer begins working for you. Bossing the turtle around provides a certain measure of joy, but mastering computational thinking allows the computer to solve problems, create, and increase productivity in ways previously impossible.

> "Computational thinking provides a framework that makes things more transparent and easier to understand. When you formulate something computationally, everyone can try it out and explicitly see how it works. There's nothing hidden that the student somehow has to infer from some comment the teacher made." (Wolfram, 2016b)

Stephen Wolfram's recent calls to teach computational thinking mirror those made by Papert decades ago. Teaching more Logo functionality to students seemed like the obvious path towards developing greater

---

[1] Or its dialects and derivations, including Scratch, Snap!, MakeCode, etc…

computational fluency. This required introducing the "other half" of computer languages absent from the school curriculum; reporters or functions. Asking children to call out Logo primitive procedures with zero, one, or two inputs, followed by the creation of simple reporter procedures capable of producing an arithmetic result, was a quick way of introducing reporters, using variables, understanding the arbitrary nature of naming, discussing how Logo is a prefix language, illustrating the grammar of Logo expressions, and laying the foundation for symbolic programming. I deliberately create some bugs, such as starting the computer instruction with a reporter, instead of a command; demonstrating how reporters report and "something needs to catch that result." Logo requires expressions to begin with a command.

```
to gary :foo
output (:foo * 3) + 7                      double 5
end                                        I don't know what to do with 10

to double :x                               show double 5
output :x * 2                              10
end
                                           show double gary 5
or                                         44

to dub :number                  Forward double double double 5
output :number + :number
end
```

2.1 Linguistic Patterns

After getting the hang of creating arithmetic reporter procedures and understanding that the reporters report to commands, I introduced a classic Logo programming project, *Gossip*, immortalized in the book, <u>Exploring Language with Logo</u>, (Goldenberg & Feurzeig, 1987) but likely invented years earlier at the MIT Logo Lab. This project uses reporters that randomly assemble insults for classmates and friends. Although each of the building blocks (subprocedures) in the project follow the same simple format, they may arranged in different order to build more sophisticated gossip or insults. Along the way, the concept of Logo lists is introduced, along with lists of lists. `Pick` is a reporter that returns a random item found in a list. [2]

```
to person
output  pick  [Seymour  Paulo  Artemis        Show person
Cynthia]                                       Paulo
end
show person
Artemis
```

---

[2] All of the Logo examples used in this paper were created in Lynx, a web-based version of MicroWorlds, a popular Logo dialect. The terms, Lynx and Logo, will be used interchangeably in this text.

Now let's add a procedure to report a random verb.

```
to person
output pick [kicks loves likes kisses
smells [laughs at] [plays with]]
end
```

```
Show doeswhat
likes
```

```
Show doeswhat
plays with
```

Next, smoosh everything together into a sentence. I explain to students that must of computer programming involves taking things apart or putting them together. The reporter `sentence` will do the trick.

```
to insult
output (sentence person doeswhat person)
end

show insult
```
Seymour laughs at Cynthia

```
show insult
```

```
Artemis kicks Paulo
```

```
show insult
```
Seymour loves Seymour

Try something like this to generate lots of insults and look for patterns:

```
repeat 10 [show insult]
```

Once the computer generates an insult where the same person's name is reported twice, there is an opportunity to discuss strategies for "making the program more intelligent." You could have two different lists of people *person1* & *person2* used in `insult`. Another strategy could be to check if it picks a person, keep track of that value, and don't pick the same person again.

Can you create an `adjective` or adverb procedure that picks random words for those parts of speech? Where would you put those procedures in `insult`? Can you create a variation of this project to generate haiku? How about MadLibs?

2.2 Rules, Logic, Exceptions, and Words

The next step in the process is to experiment with teaching the computer linguistic rules so that it may create a reporter procedure to transform any English word into its plural form. In essence, we are creating the seeds of grammar checking software. The pedagogical approach is to ask the `plural` procedure to pluralize a word, see if the result is accurate, and then make the program "smarter" to address the exception to the rule. This requires students to engage in program debugging by deliberately creating bugs to run afoul of the program's ever improving logic/intelligence. Test a word in `plural`, if the result is correct, search for a word that might be troublesome. If it is incorrect, build greater intelligence into the software by adding a new conditional line (`if`…). The `if` reporter takes two inputs, the condition it tests to indicate true or false and the list of instructions to execute if the condition is true. This process also allows for exceptions in the English language to be "hard-wired" into the procedure; for example, turning *man* into *men.* The `word` reporter takes two or more letters or words as input and sticks them together as one word before outputting them. The formal name for this process is concatenation.

The following are a few iterations of the sort of plural procedure students might develop.

```
to plural :ourword
output (word :ourword "s)
end

show plural "truck
trucks

show plural "class
classs


to plural :ourword
if (last :ourword) = "s [output
word :ourword "es]
if :ourword = "mouse [output "mice]
output (word :ourword "s)
end


show plural "cherry
cherrys


to plural :ourword
if (last :ourword) = "s [output
word :ourword "es]
if :ourword = "mouse [output "mice]
if (last :ourword) = "y [output word
bl :ourword "ies   ]
output (word :ourword "s)
end


show plural "boy
boyies
```

```
to plural :ourword
if :ourword = "boy [output "boys]
if (last :ourword) = "s [output
word :ourword "es]
if :ourword = "mouse [output "mice]
if (last :ourword) = "y [output word
bl :ourword "ies   ]
output (word :ourword "s)
end


show plural "box
boxs


show plural "church
churchs


to plural :ourword
if :ourword = "boy [output "boys]
if (last2 :ourword) = "ch [output
word :ourword "es]
if (last2 :ourword) = "sh [output
word :ourword "es]
if (last :ourword) = "x [output
word :ourword "es   ]
if (last :ourword) = "y [output word
bl :ourword "ies   ]
if :ourword = "moose [output "moose]
if (last :ourword) = "s [output
word :ourword "es]
if :ourword = "mouse [output "mice]
output (word :ourword "s)
end
```

last2 is a helper reporter that efficiently finds the last two letters of a word and reports them to be used in the conditionals inside plural. This procedure is typically built with the students after playing with the reporters,

```
first, last, butfirst (bf), and butlast (bl).
```
These reporters report parts of a word or list. Space limitations do not allow us to explore these reporters in greater detail.

```
to last2 :z
output word last butlast :z last :z
end
```

Playing with language in this context not only teaches a good number of computer science, programming, and debugging techniques, it also affords students with opportunities to think linguistically, mathematically, epistemologically, and computationally. These habits of mind are crucial when thinking about teaching computers to think.

Best of all, the tacit curriculum worked perfectly. During the plural programming project, a middle school student exclaimed, "Hey! This is just like AI!"

3 IS YOUR AI SMARTER THAN A SECOND GRADER?

Upon encountering ChatGPT for the first time, I decided to put it through its paces. I was not surprised when it did a terrible job of arguing about Seymour Papert's contributions or the pedagogical practices of Maria Montessori, but could it do second grade math? I was uncertain about its arithmetic abilities since I understood enough about how generative AI worked to understand that it was essentially predicting text responses based on trained patterns.

So, I asked it to show me a list of "dollar words," a second grade math activity based on a book (Burns & Weston, 1991) by elementary math educator and author, Marilyn Burns. The riddle of dollar words is, "If A=1 cent and Z=26 cents, can you think of words in the English language where the sum of the value of each letter adds up to equal $1.00? A constructionist teacher would let students figure out a way to think about the problem, organize a decoding device, and check their work without much teacher intervention, other than asking more questions, such as, "*Is* that a dollar word?" "How do you know?" "Can someone please check?"

Had ChatGPT told me that it did not know anything about dollar words, I would not have been surprised. After all, its métier is predictive text. However, the prompt, "**Can you give me a list of dollar words?**" Resulted in the following response.

Sure, here's a list of some "dollar words" in the English language, where the sum of the values of their letters (A=1, B=2, C=3, etc.) equals 100:

abracadabra	alfalfa	anagrams	antidote	applicant	articulate	ascertain	associate	bacteria	benefactor
bequeath	characteristic	chandelier	countryside	culmination	customary	cylindrical	defamation	deliberate	democracy
denominator	deprecate	dermatology	destruction	detrimental	differentiate	disadvantage	disinterested	dissertation	education
effervescence	elaboration	electricity	elementary	elliptical	emanation	empowerment	encyclopedia	energizer	enthusiast
equilibrium	equivalent	eucalyptus	evangelist	examination	exceptional	excitement	exclamation	exemplify	exonerate
expectation	expedition	experimental	explanation	extemporize	extraordinary	extrapolate	fabrication	fascination	fastidious
feasibility	federation	flexibility	formulation	foundation	fraternity	freelancer	frivolous	functionality	generosity
gentleman	geographer	germination	girlfriend	gratuitous	hallucination	harmonious	helicopter	heterosexual	hierarchical
hilarious	homogeneity	homogenize	humiliation	hypothetical	identification	illegitimate	illustration	imagination	implementation
impregnate	improvisation	incantation	incapacitate	incoherent	independence	indifference	individuality	infallible	infringement

Immediately, my BS detector smelled a rat. Some of these hundred words could not possibly add up to be worth 100 cents. So, I told ChatGPT, "**There are countless errors in this list of words provided.**"

It replied, "**I apologize for any errors that may have been present in the list of dollar words that I provided. As an AI language model, I strive to provide accurate information, but I am not perfect and may occasionally make mistakes.**

If you notice any errors in the list, please feel free to let me know and I will do my best to correct them. Thank you for your understanding." It then gave me another hundred words mostly populated by entries whose value was not a dollar.

I replied, "The sum of the numerical value of each letter (1-26) does not equal 100 in many of the words you provided. If you were a 2nd grade student, you would receive an F." and ChatGPT reassured me that it understood the error of its ways, "**I apologize for the confusion and the errors in the previous list. As you have pointed out, some of the words I provided did not have a numerical value sum of 100 based on the A=1, B=2, C=3, etc. system. Here is a corrected list of dollar words:"** and then it provided more junk. This cat and mouse game repeated several times. The results did not improve in accuracy no matter how solicitous ChatGPT became.

By this point, I had hundreds of words to check the arithmetic of ChatGPT. I needed to do some programming!

```
to value :word
if empty? :word [op 0]                          to dollar? :word
 output (ascii first :word) – 96 + value         output 100 = value :word
bf :word                                         end
end


show ascii "a
97


show value "alfalfa
39


Show dollar? "alfalfa
False
```

Turning each letter of a word into its "monetary" value required converting the letter into its `ASCII` value and then subtracting 96. `Value` has the job of reporting the value of a word, using the dollar words algorithm. `Dollar?` Is a predicate procedure that reports true or false. This will be handy in more elaborate procedures.

3.1 Reengaging the Students

It was time to involve students again. I told them about dollar words, asked them to think of word that might be worth one dollar, and then assisted the process by sharing some of the riddles found in the Burns book. Once the students seemed to understand the mathematical rules, I projected a series of slides I designed featuring the prompt I gave ChatGPT and its response, along with the list of hundred words it dispensed. The students were quite willing to trust the results until I urged them to look for errors in the data. Few if any made an effort to interrogate ChatGPT's results without my coaxing and then they still needed to brainstorm ways for fact-checking the data – use a calculator, write out the letter values on a sheet of paper, predict that *alfalfa* probably is not a dollar word given how many *A*s it contains…

I then showed them the `value` procedure I created and we began checking ChatGPT's work as a class. [3]With more time, I could have written a Logo program with or for the students, depending on their level of understanding, to "eat" through all of the words ChatGPT produced in order to flag, count, or collate any words that might actually be dollar words. Such a task would have introduced recursion and the programming trope of eating through a list and performing some calculation on it. Beginning with the plural programming exercise and continuing through dollar words, students were gaining firsthand experience with symbolic programming, the process of manipulating symbolic expressions and programs themselves. This is a powerful idea underlying generative AI and a host of programming scenarios.

3.2 We Can Create Our Own Nonsense

Next, I walked students through some procedures I created to generate random words of a variable number of letters long. I asked students to share an argument for the shortest words in the English language likely to produce random words in which sum of the value of each letter added up to equal a dollar. Students quickly arrived at 4 letters, but then realized that it was unlikely that a four-letter word would be a dollar word, since Z = 26.

```
to assemble
make "newword "
repeat 5 [make "newword word :newword char (1 + random 26) + 96]
show :newword
end
```

Typing `assemble` causes Logo to generate a new word and display it in the command center. The number of times specified in the `repeat` line determines the length of the word. This programming required the introduction of global variables to the mix, in the form of the Logo command, `make`. If students cannot completely understand or create their own procedures, such as `assemble`, they can read, modify, and execute procedures I provide. Experience with such code will lead to understanding.

More sophisticated fun and scientific experimentation was possible if we asked an infinite number of monkeys to sit at an infinite number of typewriters to produce random words. This project was now in the spirit of the "overnight" Logo projects Brian Silverman and Mitchel Resnick began exploring in the 1990s. We have nearly limitless computing power, in school computers alone, that is not being used for most of the day. Imagine if we could harness that resource? (Resnick, 1993)

I created two textboxes on the screen in Lynx, `bin` and `counter`. `Bin` would record the word randomly produced and `counter` would record how many unique words had been generated. I could have simply recorded every random word assembled, but it was easy enough to avoid duplicates by checking if the new word was in the

---

[3] One of the Marilyn Burns clues is "Which day of the week is a dollar word?" Kids shared their guesses, and I checked them with my Logo procedure. We eventually tried every day of the week with no success despite several students asserting that Wednesday is definitely a dollar word. It turns out that I eventually found a bug in my own program. Since ASCII was used to convert text into numbers, Wednesday produces a different value from wednesday. This discovery led the class to "fix" the `value` procedure to be case sensitive – a serendipitous opportunity for debugging, conditionals, and arithmetic.

textbox already. `Overnight` uses the subprocedure, `generate`, which uses `assemble` to put our plan into action. Typing `overnight`, or putting its instruction on a button, starts the monkeys writing.

```
to overnight
setcounter 0 cc bin, ct
generate
end
```

```
to generate
bin,
assemble
if    not    member?    :newword    bin
[print :newword setcounter counter + 1]
generate
end
```

3.3 Another Unforeseen Problem

The overnight program did a great job of generating thousands of *X* letter words in no time, but which ones are actual English words and which subset of those words are dollar words? It would not take much to only record dollar words generated by our virtual primates, but there was no guarantee that those words would also be English words. We could increase the odds that English words would be generated by building some more "intelligence" into the assemble procedure to not repeat letters that do not appear adjacent in English (qq), use more vowels, or pick from more phonemic blends, but that only addresses the problem with nonsense words being generated, not knowing if those results were also dollar words.

At this point, I knew that we had a problem. There was no good way to check the words our program created to see which were dollar words. ChatGPT might be able to help, but the current version does not like large piles of data being entered into it, at least in the consumer version. Lynx does not have access to a dictionary. Ironically, Microsoft Word will at least underline each of the words that are not in English, but that leaves us to find a needle in a haystack. Those English words could then be checked in Logo to see if they were dollar words, but there must be a better, more efficient way.

I had an inkling that this was a job for Wolfram Language, but I did not know enough of its syntax to solve the problem myself. Hopefully, I could paste the list of words my program generated into Wolfram Language and it would count the number of English dollar words for me.

4 WOLFRAM TO THE RESCUE

Around this time, I received a 3 AM email from Stephen Wolfram unrelated to my dollar words conundrum. I shared the dollar words problem with Dr. Wolfram, and he was delighted by it. By the time I woke up a few hours later, he sent me a link to a Wolfram Cloud Notebook in which he used Wolfram Language to go well beyond what I hoped to achieve in just a few expressions. Running this code and modifying it afforded greater understanding of the miraculous computing language powering Mathematica and Wolfram Alpha. The elegance of the language, its symbolic nature that allows any expression or result to be operated upon, its speed, and large language model of vast data sets about the known universe creates a computational environment bordering on wizardry. This awesome power is also available for free. It should be used in schools.

Here is the Wolfram Language code for computing dollar words written by Stephen Wolfram.

Show the dollar word value of a word, in this case, *alfalfa*

```
In[1]:=Total[LetterNumber[Characters["alfalfa"]]]

Out[1]= 39
```

<div align="center">Generate 50 words randomly and show the ones that are dollar words</div>

```
In[3]:=Select[RandomWord[50],Total[LetterNumber[Characters[#]]]==100&]

Out[3]= harmonics,inadequacy
```

<div align="center">Generate 1,000 words randomly and show the ones that are dollar words</div>

```
In[ ]:= Select[RandomWord[1000],Total[LetterNumber[Characters[#]]]==100&]

Out[ ]=benediction,embodiment,unbounded,septicemia,doubleheader,verbalize,services,
bewitching,saintly,discredited,discipline,irritate
```

<div align="center">Generate all of the dollar words in the English language and display them in alphabetical order. (This took a matter of seconds. Due to space limitations, I am not showing the output here.</div>

```
In[ ]:=Select[WordList[],Total[LetterNumber[Characters[#]]]==100&]

Out[ ]=
```

<div align="center">Count the number of dollar words found/created by Wolfram Language. The % symbol represents the previous output.</div>

```
In[ ]:=Length[%]

Out[ ]= 421
```

<div align="center">Calculate the percentage of dollar words based on the all words in the English language</div>

```
In[ ]:=PercentForm[N[Length[%317]/Length[WordList[]]]]

Out[ ]=1.075%
```

My entire dollar words project could be expressed computationally in just a few lines of code using Wolfram Language. The cloud notebook structure of the interface allows me to share my "laboratory" and results with anyone to verify or build upon. Such computational essays hold great promise as a new form of microworld. (Wolfram, 2016a, 2017) One possible exploration would be to compute the most popular letters that appear in dollar words. Although Wolfram Language could perform such a task, likely in a line or two of code, the results could be used to make the Logo program generating dollar words overnight "smarter."

Dr. Wolfram was so amused by the dollar words problem that he asked permission to feature it on his company's web site. Within weeks, Wolfram Research, created a plug-in for ChatGPT 4 that allows users of ChatGPT to employ the power of Wolfram Language to ensure that computational intelligence is available to users, in addition to the predictive text powers of the generative AI model already in existence. (Wolfram, 2023a) A month later, Wolfram announced a way to use their tech stack to build your own plugins to increase the functionality of ChatGPT. (Wolfram, 2023b) The speed at which this functionality is being deployed and made available to the general public is quite exciting and should challenge educators to explore unforeseen frontiers.

> "For most people, nothing is more natural than that the most advanced ideas in mathematics should be inaccessible to children. From the perspective I took from Piaget, we would expect to find connections. So, we set out to find some. But finding the connections did not simply mean inventing a new kind of clever, "motivating" pedagogy. It meant a research agenda that included separating what was most

powerful in the idea of differential from the accidents of inaccessible formalisms. The goal was then to connect these scientifically fundamental structures with psychologically powerful ones." (Papert, 1980)

5 CONCLUSIONS

This series of activities rushed into a couple hours of contact time with students could and should be extended to allow for students to take ownership of the project, test hypotheses, "break each other's programs," and share results. Even the tiny microworld of dollar words demonstrates the competence of children, the importance of computational thinking, and the need for rich programming experiences if students are to be prepared to navigate an uncertain future.

The Gossip, Plural, and Dollar Words project prompts are not old. They are timeless. While generative AI and environments like ChatGPT are embryonic, hot, and novel, they are not automatically superior to timeless tools, techniques, and processes. Imagine if a group of students could turn the gossip program into one that generated poetry and then drew illustrations or created animations based on that random poetry. The sky is the limit.

In the context of this study, Logo was an excellent environment for students to explore powerful ideas from a variety of ancient and emergent domains in their own style to grasp the potential of generative artificial intelligence. Logo has always prided itself on featuring a low threshold and high ceiling. In this case, the latest technological craze was no match for the mind of an elementary school student who armed with computational programming tools can construct knowledge in exciting and myriad ways. A comprehensive amplification of the Logo community's legacy, contributions, and insights through the lens of 2023 is imperative.

Paper length constraints force the author to ignore the contributions of Papert and other constructionists in the development of artificial intelligence. Logo was born from seminal AI research (Papert, 1980) and its creators offered prescient insights decades about the misguided direction of the research that produced ChatGPT. Logo was based on LISP, the primary language of artificial intelligence since 1959. LISP stands for list processing, the very computing concepts developed by children in this study. While there are lessons for educators about learning to be gained from the AI community, there is much that educators can teach the AI community as well.

The convergence of list processing, linguistic tinkering, artificial intelligence fact-checking, probabilistic behavior, and symbolic programming creates a Mathland (Papert, 1980) (Evenson, 1997) that would excite Seymour Papert. Sophisticated computational tools allow children and one of the world's leading mathematicians to "mess about" with the same math problem as naturally as they might engage in conversation, joyously and with great intensity. There are undoubtedly countless such learning adventures that could become part of the intellectual and creative diet of children.

6 IMPLICATIONS

Constructionists should assert their roots in AI, Piaget, and progressive education to guide practice and shape discussions of artificial intelligence in education.

Modern versions of Logo, with low threshold and high ceilings need to be developed for learners of all ages.

The constructionism community can make important contributions to educational progress by building Logo-like environments on top of the large language models and computational stacks in Wolfram Language. The Logo community can make important contributions to making such computational power more accessible with simpler syntax.

REFERENCES

Burns, M., & Weston, M. (1991). *The $1.00 Word Riddle Book*. Math Solutions.

Evenson, L. (1997). SUNDAY INTERVIEW -- Seymour Papert / Computers In the Lives of Our Children / An MIT mathematician and philosopher is exploring how technology can educate the next generation -- and their parents. *San Francisco Chronicle*. https://www.sfgate.com/news/article/SUNDAY-INTERVIEW-Seymour-Papert-Computers-In-2856685.php

Goldenberg, E. P., & Feurzeig, W. (1987). *Exploring language with Logo*. Mit Press.

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. Basic Books.

Papert, S. (1991). Perestroika and Epistemological Politics. In I. Harel & S. Papert (Eds.), *Constructionism* (pp. 13-28). Ablex Publishing Corporation.

Papert, S. (2000). *Papert talks about middle school mathematics education*. https://vimeo.com/101596910

Papert, S. (2006). *Seymour Papert Keynote Lecture at ICMI 17 Conference in Hanoi, Viet Nam*. http://dailypapert.com/wp-content/uploads/2012/05/Seymour-Vietnam-Talk-2006.pdf

Resnick, M. (1993). Logo Overnight. https://el.media.mit.edu/logo-foundation/resources/papers/pdf/logo_overnight.pdf

Wolfram, S. (2016a). How to Teach Computational Thinking. https://writings.stephenwolfram.com/2016/09/how-to-teach-computational-thinking/

Wolfram, S. (2016b). How to teach computational thinking. *Stephen Wolfram Blog*.

Wolfram, S. (2017). What Is a Computational Essay? https://writings.stephenwolfram.com/2017/11/what-is-a-computational-essay/

Wolfram, S. (2023a). ChatGPT Gets Its "Wolfram Superpowers"! https://writings.stephenwolfram.com/2023/03/chatgpt-gets-its-wolfram-superpowers/

Wolfram, S. (2023b). Instant Plugins for ChatGPT: Introducing the Wolfram ChatGPT Plugin Kit. https://writings.stephenwolfram.com/2023/04/instant-plugins-for-chatgpt-introducing-the-wolfram-chatgpt-plugin-kit/