International Journal of Scientific Research and Engineering Development-– Volume 4 Issue 5, Sep-Oct 2021

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Evolution Of Artificial Intelligence and Its Impact On Programming

Pratik Shrestha[1] **Kritish Pokharel**[2] **Nitiz Khanal**[3]

[1,2,3](High school graduates, St.Xavier's College Maitighar)

[1](shresthapratik74@gmail.com)

[2](kritish.pokharel@gmail.com)

[3](khanalnitij20@gmail.com)

----------**********----------

ABSTRACT

The recent release of Codex, a GPT based language model commonly referred to as a descendent of the GPT-3 model, has caused quite a stir in machine learning and programming communities. GitHub Copilot is powered by the distinct production version of the model. Its training data includes both natural language and billions of lines of source code from publicly available sources, such as code in public GitHub repositories. While Python is OpenAI Codex's strongest suit, it is also skilled in a wide range of other languages, including JavaScript and Go as well as PHP and Ruby. Compared to GPT-3, which has just 4KB of RAM for Python code, it can take into consideration almost three times as much contextual information when executing any task. In layman's terms, the GitHub copilot may be viewed as a code autocompletion tool. A function name, its description, and a few other details are entered, and it produces the code for you quite correctly! In addition, it is capable of generating considerably larger and more comprehensive functions than traditional autocompletion tools do. As machine learning and natural language processing models continue to develop, this research analyzes what the future of programming will look like with these continual advances in technology. As a starting point, this article will discuss the history of artificial intelligence and how it has dominated the IT sector in the last decades. A brief introduction of Natural Language Processing and its influence on computer science will be followed. Then we will examine Artificial Intelligence and Natural Language Processing models like Codex, GPT-3, RNN, Perceptron, etc as well as the underlying technology that powers them. The impact of these technical breakthroughs in programming will also be discussed, along with its potential for the future. Thus, we will address how programmers should be aware of these developments in Artificial Intelligence and should restructure their programming talents to keep up with this quick growth and prepare themselves for future needs.

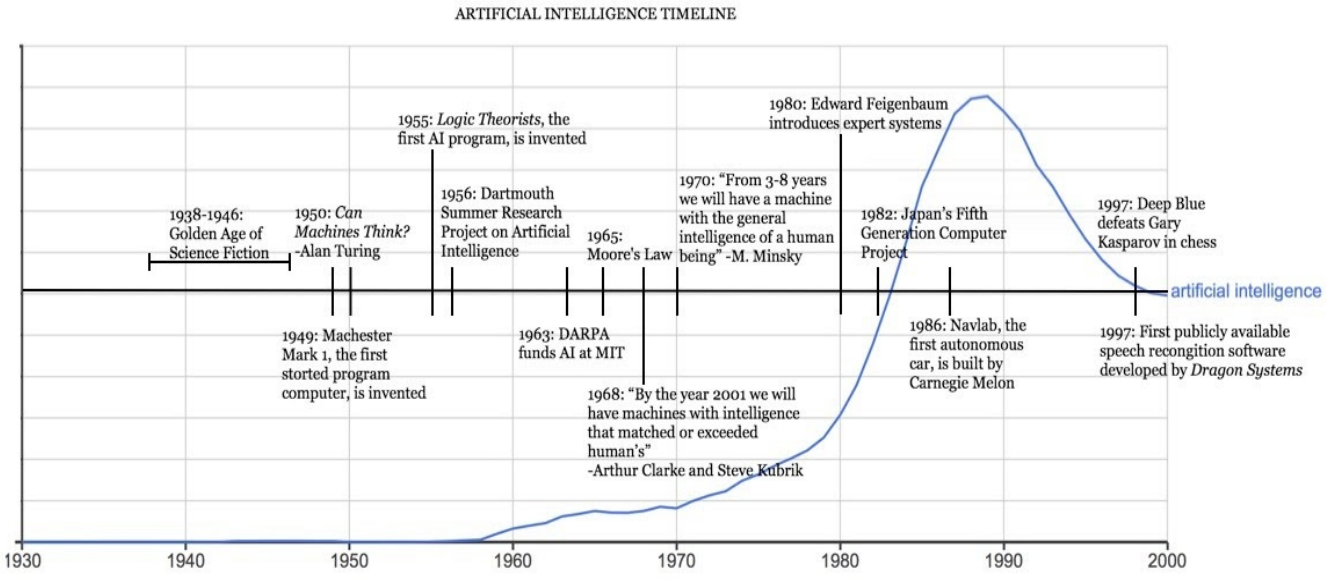
Keywords**: Copilot** , **GPT** , **Codex** , **RNN**, **NLP** , **OpenAI**.

----------**********----------

I. INTRODUCTION

History Of Artificial Intelligence

Back in the 1950s great scientists, mathematicians and philosophers started discussing artificial intelligence. One such was Alan Turing, a British mathematician who suggested that humans use available information as well as a reason to make decisions, so why can't machines do the same thing? This was the logical framework of his 1950 paper, Computing Machinery and Intelligence in which he discussed how to build intelligent machines and how to test their intelligence[1]. But back then computers weren't as advanced as they are today. They lacked prerequisites for intelligence such as they could execute the commands but not remember what they executed. Also, the cost of hiring computers at that time was too high and only some rich universities could afford it. Things changed in 1955 when the first artificial intelligence *program* (the first program specially engineered to mimic the problem-solving skills of a human being) was created in 1955-56 by Herbert Simon, Allen Newell, and John Shaw[2] and was presented at Dartmouth Summer Research Project on Artificial Intelligence in 1956. At this conference, researchers from different fields gathered together to talk about artificial intelligence, and the program was funded by Research and Development (RAND) Corporation. Though the event fell short of expectations, it catalyzed the next 20 years of AI research. From 1957 to 1974 AI flourished, computers became powerful and machine learning algorithms became better. Not only that, but people also started knowing which algorithm to use to solve their problems. Government agencies such as the Defense Advanced Research Projects Agency (DARPA) fund AI research at several institutions[3]. The government was particularly interested in a machine that could transcribe and translate spoken language as well as high throughput data processing. The diagram shows the major accomplishments in AI between 1938 to the end of the 20th Century.



. Fig 1: Timeline of progress in AI between 1938-2000[1]

The end of the 20th century and the start of the 21st century is known as the golden age for AI as during the 2010s, humans saw the biggest achievement in AI when in 2011 IBM Watson won Jeopardy beating former champion Brad Rutter and Ken Jennings[4]. The impact of deep learning in the industry also began in the early 2000s, when CNNs already processed an estimated 10% to 20% of all the checks written in the US, according to Yann LeCun Industrial applications of deep learning to large-scale speech recognition started around 2010[5]. Similarly in 2012 google's brain-computer cluster trains itself to recognize cats from millions of images in youtube videos and technologies like Apple Siri(2011) and Microsoft Cortana started using NLP to answer questions and perform actions. Robot HRP-2 built by SCHAFT Inc of Japan, a subsidiary of Google, defeated 15 teams to win DARPA's Robotics Challenge Trials. HRP-2 scored 27 out of 32 points in 8 tasks needed in disaster response. Tasks were driving a vehicle, walking over debris, climbing a ladder, removing debris, walking through doors, cutting through a wall, closing valves, and connecting a hose[6]. The advancement in AI on one side was considered a boon for humans but it also created fear of being misused against the same human race. As a result, an open letter to ban the development and use of autonomous weapons was signed by Hawking, Musk, Wozniak, and 3,000 researchers in AI and robotics[7]. Following this, Asilomar Conference on Beneficial AI was held to discuss AI ethics and how to bring about beneficial AI while avoiding the existential risk from artificial general intelligence.AI in gaming was a talk of the town when OpenAI-machined learned bot played at The International 2017 Dota 2 tournament in August 2017. It won during a 1v1 demonstration game against professional Dota 2 player Dendi[8]. The next was followed in 2015 when Google DeepMind's AlphaGo (version: Fan) defeated 3 time European Go champion 2 dan professional Fan Hui by 5 games to 0[9]. The prominent use of AI in daily life was shown in 2018 Google I/O where Google's Assistant can actually ring up a salon or a restaurant to make an appointment for you. You don't have to call yourself even if the pizzeria doesn't have an online reservation system. Open AI GPT-3 was presented in 2020 coupled with the launch of GitHub co-pilot in 2021, showing the AI progress towards automated programming.

Natural Language Processing

From the art of understanding each other to making computers comprehend our natural languages, we have come a long way. Natural Language Processing (NLP) is concerned with the interplay of natural language and computers, as the name indicates. It is a key component of computational linguistics and Artificial Intelligence (AI). It allows computers and humans to connect in an efficient way, and it uses machine learning to enable computers to interpret human speech.

For numerous reasons, human language is unique. It was created with the intent of conveying the speaker's or writer's message. It's a complicated system, yet somehow small toddlers can pick it up fast. Due to its complexity, comprehending human language is considered a tough endeavor. There are an endless number of ways to order words in a phrase. Furthermore, words might have several meanings, necessitating the use of contextual information in order to accurately comprehend sentences. Every language is different and confusing in some way. An AI that can analyze all of the information accessible on the internet, resulting in artificial general intelligence, would be the outcome of a flawless understanding of language by a machine.

Natural Language Processing is basically divided into two categories: Natural Language Understanding and Natural Language Generation.[10]

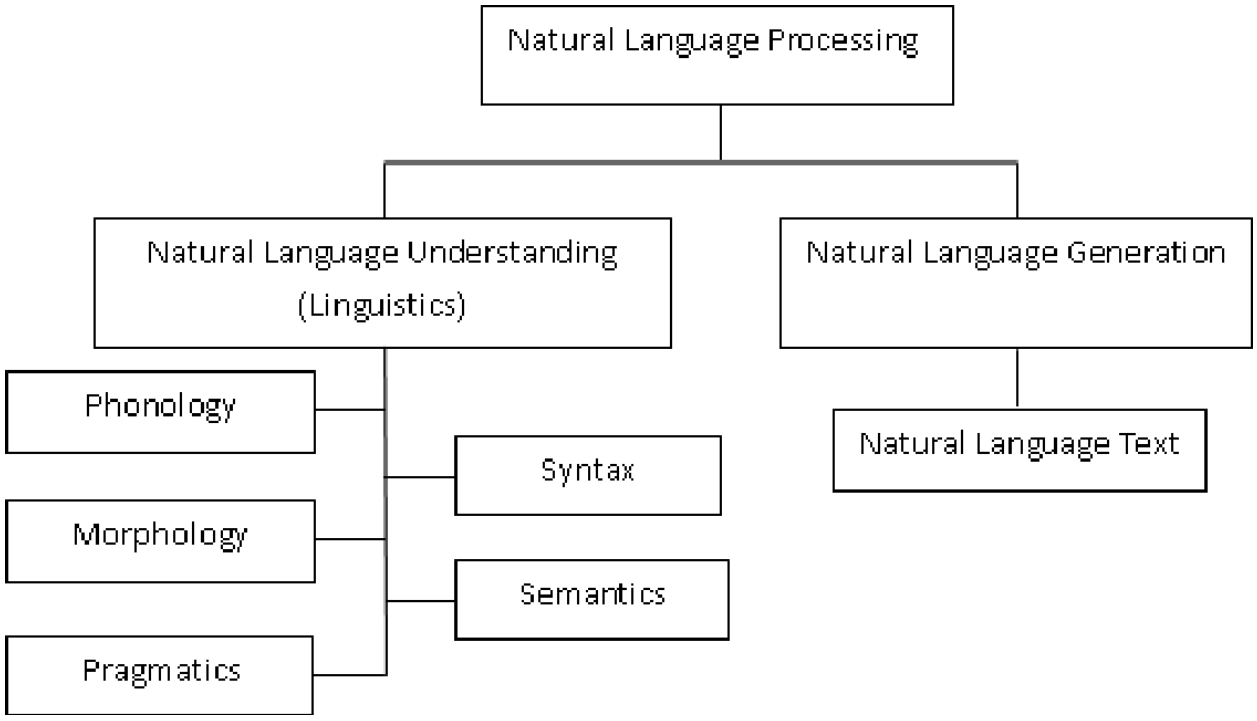


Fig 2: Basic Classification of NLP[10]

Natural Language Understanding eases human-computer interaction. The ability of computers to interpret commands without the codified syntax of computer languages is due to their knowledge of human languages such as English, Spanish, and French. NLU also allows computers to converse with people in their native tongue.NLU analyzes data to identify its meaning by reducing human speech to a structured ontology, which is a data model made up of semantic and pragmatic concepts. Intent and entity recognition are two key elements in NLU. The technique of recognizing the user's emotion in the input text and establishing their goal is known as intent recognition. Because it defines the meaning of the text, it is the first and most essential component of NLU. Entity recognition is a subset of natural language understanding that focuses on detecting the entities in a message and extracting the most significant information about those entities. Named entities and numeric entities are the two sorts of entities. Natural Language Understanding comprises Phonology (sound), Morphology (word creation), Syntax (sentence structure), Semantics (syntax), and Pragmatics (understanding).

The two major approaches for interpreting natural language are syntactic analysis (syntax) and semantic analysis (semantic). Language is a collection of valid sentences, but what constitutes a legitimate sentence? Syntax and semantics are two different things. The grammatical structure of the text is known as the syntax, whilst the meaning communicated is known as semantics. A syntactically valid statement, on the other hand, is not always semantically accurate. “Goats flow phenomenally,” for example, is technically correct (subject-verb -adverb), but it makes no sense. Syntactic analysis, often known as syntax analysis or parsing, is the practice of using formal grammar principles to analyze natural language. Individual words are not subject to grammatical rules; instead, categories and groupings of words are. Syntactic analysis is the process of assigning a semantic structure to a piece of text. Similarly, semantic analysis is the act of deciphering and interpreting the meaning and structure of words, signs, and sentences. This allows computers to interpret natural language similarly to humans. Partially because semantic analysis is one of the most difficult aspects of NLP to master, and it is yet unsolved.

Phonology is a branch of linguistics that deals with the systematic organization of sound. Phonology, according to Nikolai Trubetzkoy in 1993, is "the study of sound relating to the system of language." Whereas Lass stated in 1998 that phonology is concerned with the sounds of language and is a sub-discipline of linguistics and that it may be defined as follows: "The function, behavior, and structure of sounds as linguistic objects are all addressed by phonology. The semantic use of sound to encode meaning in any Human language is referred to as phonology.[11]

Morphology is the study of word structure and meaning. Some words, like send, appear to be atomic or monomorphemic, whereas others, like sends, sending, and resend, appear to be made up of several atoms or morphemes. These ‘word fragments' are morphemes because they appear often in other words - thoughts, thinking, reprogram, rethink. The way morphemes combine has a syntax - the affixes discussed so far all join with verbs to form verbs, whereas others, such as ability, combine with verbs to form adjectives – programmable – and so on. [12]

Natural Language Generation (NLG) is the process of generating meaningful phrases, sentences, and paragraphs from an internal representation. It is a component of Natural Language Processing and consists of four stages: defining objectives, planning how goals may be attained by analyzing the circumstances and accessible communication sources, and implementing the plans as a text. It is the opposite of understanding.

Alan Turing published a paper in 1950 outlining a test for a "thinking" machine. He said that if a computer could participate in a discussion using a teleprinter and resemble a person so well that no discernible deviations could be detected, the machine might be regarded as capable of thinking. The Hodgkin-Huxley model, published in 1952, demonstrated how the brain utilizes neurons to build an electrical network.

Natural Language Processing and Artificial Intelligence have revolutionized the computer and programming industries, from Alan Turing's dream of a thinking machine to Microsoft's release of the GitHub copilot. The most common applications of Natural Language Processing include machine translation, spam filtering, text classification, information extraction summarization, and so on. Models such as BERT, XLNet, RoBERTa GPT-2, GPT-3, and now codex are

reinventing programming and language processing. GitHub Copilot, a code autocompletion tool, powered by the recently launched Codex, which is built on top of GPT-3, where GPT is known as the most dangerous AI model series. The further sections will provide a thorough examination of these mind-blowing models and their emergence.

II. JOURNEY TO GitHub CO-PILOT

NLP models are more accurate than ever before, thanks to the exponential growth in the text data available on the internet. A wide range of NLP models has been developed over time[16][17][24][25][26][30][32], experimenting on what works and what does not. In this section, the development of these models is discussed leading up to state-of-the-art models like GPT-3, Codex**.**

Neural network models operate on numerical data. So, it cannot directly take text data as input. The text data should first be represented in data of numeric form. One of the methods to do this is One Hot Encoding[13]. For this, a vocabulary of a certain number of words is considered. The word not in the vocabulary can be represented as <UNK>. A word is represented by a vector of size (vocabulary size X 1), with 0 everywhere and 1 in the position where the word is located in the vocabulary. For instance, consider a vocabulary V= [‘a’, ‘ant’, ‘ball’, ‘cat’, ..., ‘zebra’, <UNK>]. With reference to vocabulary V, the word ‘ball’ can be represented as a vector [0, 0, 1, 0, ……., 0, 0]. So, sentence X can be represented by a sequence of one-hot vectors [$x^{<1>}$, $x^{<2>}$,.....,$x^{<Tx>}$], where T_x is the length of the sentence.

One Hot Encoding converts the word into vectors which can be processed by NLP models. But it misses other inherent information in a word. For eg: a girl, a boy gives information about gender, Kathmandu, Washington D.C is the capital city of a country. This information cannot be represented in One hot Encoding alone because One Hot Encoding just encodes words in numeric form based on its position on vocabulary. For this task, algorithms like (2013)Word2Vec [14], (2014)GloVe [15] are used . In this algorithm, an embedding matrix is learned using a single-layer neural network. The matrix encodes possible kinds of information the word may convey. Eg, gender, name of the city, name of a fruit. The word is multiplied by the embedding matrix to receive a richer representation.

Working on sequences of data like text is more challenging than numerical data. Regular Neural Networks do not do well on these kinds of data because of the linkage of data in sequence data. For instance, in the sentence “The cat is sleeping”, the verb ‘is’ is dependent on the subject ‘cat’. In regular neural nets, this kind of dependency cannot be captured. Furthermore, text data is of varying length, which cannot be trained on neural networks with fixed input sizes. So, a new kind of network is required called Recurrent Neural Network(RNN)[16].

RNN consists of a network unit recurring multiple times, hence the name Recurrent Neural Network. It captures the dependency between data using a linkage between the network units. This linkage is created by passing the activation values from a unit to the next unit. Initially, an activation vector $a^{<0>}$ initialized with 0s, and the first-word $x^{<1>}$ is passed into the first unit, which outputs activation $a^{<1>}$ and optional output $\hat{y}^{<1>}$. The activation $a^{<1>}$ is then passed into the next unit, which receives $a^{<1>}$, and $x^{<2>}$ and outputs $a^{<2>}$. So, the information about the first word $x^{<1>}$ is made available to the

second unit when outputting predictions for $x^{<2>}$. In general, any unit in place t will have information about previous t-1 words. RNN is a flexible model which allows for single input multiple outputs, multiple input multiple outputs, and so on.

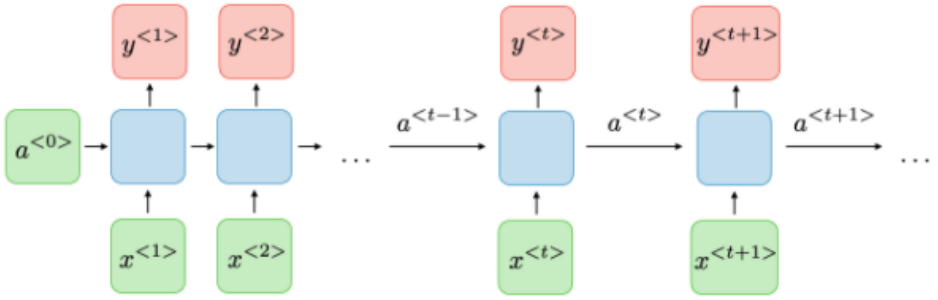


Fig 3: A simple RNN[17]

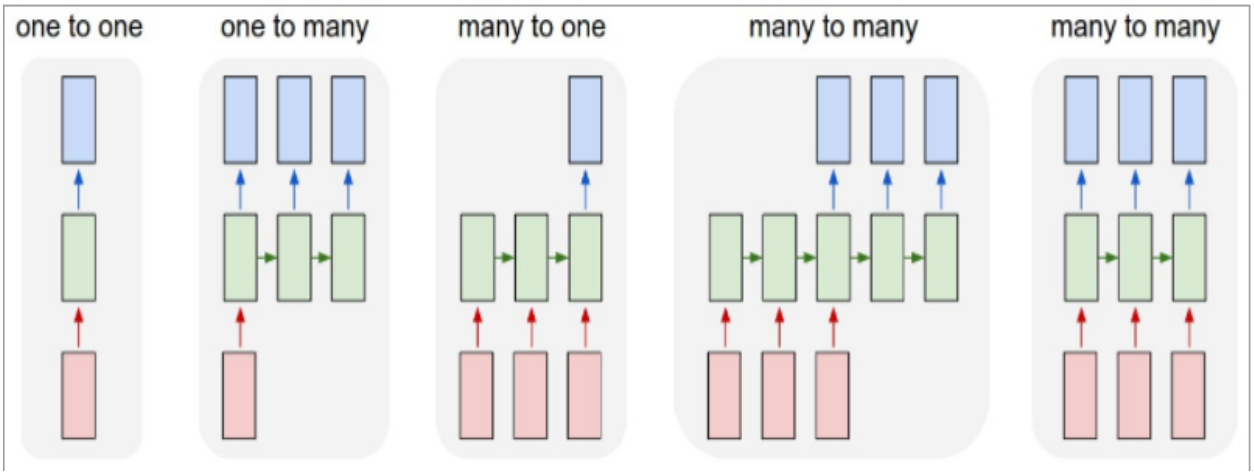


Fig 4: Different Types of RNN[18]

Though the RNN model seems to capture the dependencies between the data, it fails to do so for long-range data. As for example, in the sentence “The children, who were playing in the field, have left” the choice of the word “have” is dependent on the word “children” which is at the far right. This is due to a problem of vanishing gradients in RNN. The properties captured by activation values calculated towards the far right of the sentence, get overshadowed by the new values calculated over subsequent units and vice versa. Hence, units at the far right do not get much information about initial words. In other words, plain RNN fails to capture long-term dependencies in data.

To solve this problem of vanishing gradients, Long Short Term Memory (LSTM) [19] and Gated Recurrent Unit (GRU) [20] were introduced in 2015. They capture the long-term dependencies using memory cells. In LSTM, the memory cell is initialized to 0s or consists of random values. For an RNN unit t, a candidate $\tilde{c}^{<t>}$ is calculated for replacing the current value of the memory cell.

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

Here, W_c and b_c are learnable weights and biases associated with the memory cell. The decision of updating the memory cell is made with the help of update gate (Γ_u) and forget gate(Γ_f).

$$\Gamma_u = \text{sigmoid}(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \text{sigmoid}(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

Finally, the memory cell is updated as,

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t>}$$

Here, * denotes element wise product.

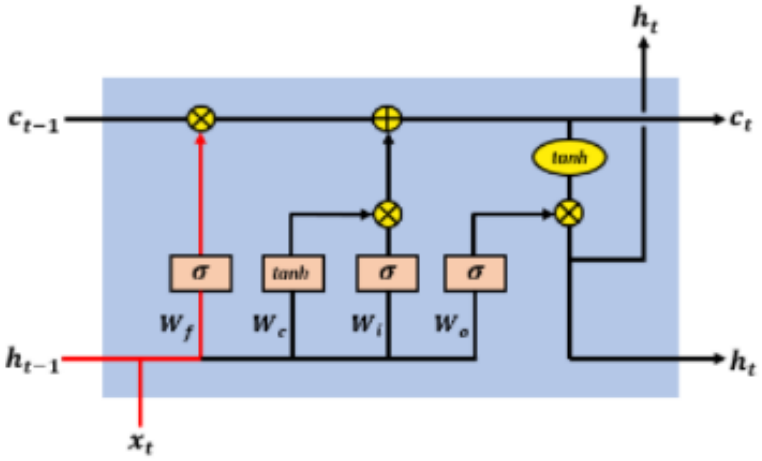


Fig 5: A LSTM unit[21]

LSTM succeeds in capturing long-term dependencies. But a unidirectional LSTM can only capture patterns from left to right since the activations are passed in that fashion. Real-world data have dependencies in both directions. In the sentence "He went there (by) car.", the choice of the preposition "by" depends on what comes later. If we add "a" in front of the car, "in" should be used. "He went there (in) a car". This bidirectional dependency can be captured with the help of bidirectional LSTM [22]. In bidirectional LSTM, two activations $a^{\rightarrow<t>}$ and $a^{\leftarrow<t>}$ are calculated. The $a^{\rightarrow<t>}$ is passed from left to right and $a^{\leftarrow<t>}$ is passed from right to left. Prediction is made after completing both forward pass and backward pass through the units. So, the final output is based on dependencies from both directions.

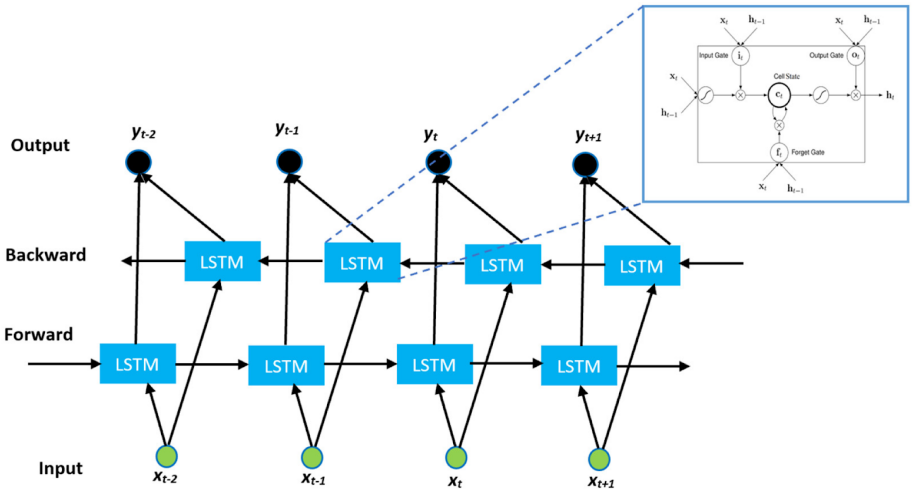


Fig 6: Bidirectional LSTM [23]

RNN and LSTMs process information sequentially. Activations from one unit are passed onto other units one by one. These steps cannot be parallelized, as a unit requires input from the previous unit to do its calculations. So, training RNNs takes a lot of time. The Google Brain research team introduced a new model "Transformer", in the paper "Attention is all you need" [24] which uses an attention-based mechanism to handle long-term dependencies. The calculation for all the words can be done simultaneously. Furthermore, there are lots of independent calculations. Hence, the transformer can be parallelized leading to reduced train time. In this model, for each word of a sentence attention-based vectors are calculated.

$$A(q, k, v) = \sum_i \frac{exp(q.k^{<j>})}{(\sum_j exp(q.k^{<j>})} . v^{<i>}$$

Here, q, k, and v are called query, key, and values. They are calculated as:

$$q^{<i>} = W^Q.x^{<i>}$$
$$k^{<i>} = W^K.x^{<i>}$$
$$v^{<i>} = W^V.x^{<i>}$$

The idea behind these values is that $q^{<i>}$ is a query about the word $x^{<i>}$, $k^{<i>}$ is the answer values to the queries $q^{<i>}$, and $v^{<i>}$ handles how these values are represented in A(q, k, v). So, A(q, k, v) is the representation of the word based on the queries $q^{<i>}$ asked to the surrounding words $x^{<i>}$s which are answered by their keys $k^{<i>}$. Therefore, it captures the context of other words of the sentence.

The above steps can be calculated simultaneously as

$$A(Q, K, V) = \text{softmax}(\frac{Q.K^T}{\sqrt{d_k}}).V$$

Where Q, K, V are a set of queries ($q^{<1>}$, $q^{<2>}$,....), keys($k^{<1>}$, $k^{<2>}$,....) and values($v^{<1>}$, $v^{<2>}$,.....).

The step of calculation of one attention vector for each word is single head attention. In a transformer, several attention values are calculated for words, and multi-head attention is calculated.

For this, the queries, keys are multiplied by different weights, corresponding to a different number of attention values to be calculated and those values are concatenated giving multi-head attention.

$$\text{Multihead attention} = \text{concat}(\text{head}_1, \text{head}_2, \ldots \text{head}_n)$$

$$\text{head}_i = \text{Attention}(W_i^Q.Q, W_i^K.K, W^V{}_i.V)$$

The position of elements are not considered in these attention values, so separate positional encoding is calculated as:

$$PE_{(pos,\ 2i)} = \sin\left(\frac{pos}{1000^{2i/d}}\right)$$

$$PE_{(pos,\ 2i+1)} = \sin\left(\frac{pos}{1000^{2i/d}}\right)$$

The transformer consists of two parts, encoder, and decoder. The encoder part consists of multi-head attention and a feed-forward neural net with normalization in each step. First, the input sentence along with the positional encoding is fed into the multi-head attention, which is then fed into a feed-forward neural network. The neural net finally gives K and V as output. Over towards the decoder block output is fed into masked multi-head attention which outputs Q. In masked multi-head attention, words towards the right are masked so that the model can generate predictions from previous words and train on the data. Q from multi-head attention along with K and V received from the encoder block is then passed onto another multi-head attention. The multi-head attention is then linked to a feed-forward neural network. Linearity is applied to received output and it is passed to a softmax layer which outputs the probabilities of next words. Normalization layers are inserted in each step. The encoder and decoder blocks are repeated N_x times before giving output.

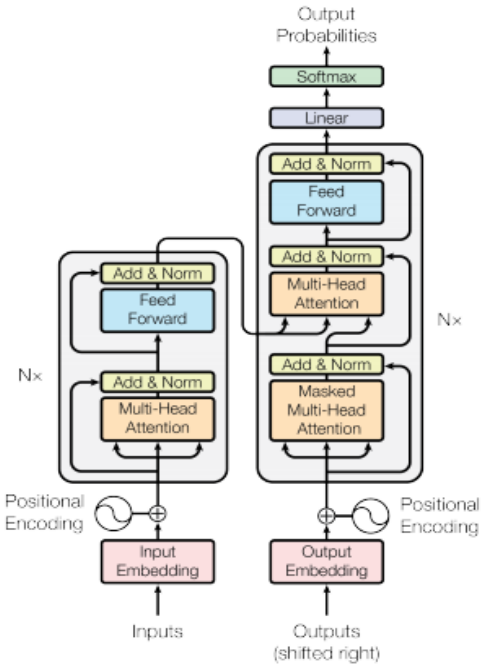


Fig 7: The Transformer model architecture [24]

RNN, LSTM, Transformers were used mostly by fine-tuning them to specific tasks. Mostly supervised learning was used, which required a labeled dataset. The models were used to learn word-level information. The Generative Pre-Training model [25] introduced by a group of researchers at Open AI in 2018 changed this practice. They used semi-supervised learning; unsupervised learning on a large corpus of unlabelled text (BookCorpus dataset: a dataset of 7000 unpublished books) and supervised learning on a labeled dataset. Unsupervised learning allows for the model to train on a humongous amount of unlabelled data. The GPT model learns all kinds of language tasks in general, which is then optimized through supervised learning. With the GPT model, researchers aimed at capturing higher-level semantics, rather than just word-level information. Semi-supervised learning along with 117M parameters of the GPT model, outperformed task-specific models on 9 out of 12 tasks the researchers studied including commonsense reasoning, question answering, textual entailment.

The unsupervised part of the GPT is performed with the decoder part of the transformer. It is repeated 12 times. For the given unlabelled data $U= [u^{<1>},u^{<2>},\ldots,u^{<n>}]$, the objective function $L_1(X)$ is optimized:

$$L_1(U) = \sum_i logP(u^{<i>}|u^{<i-k>},.....,u^{<i-1>};\Theta)$$

where k is the size of the context window, and the conditional probability P is modeled using a neural network with parameters Θ. These parameters are trained using stochastic gradient descent. Multilayer transformer is used to produce output distribution as:

$$a_l = UW_e + W_p$$
$$a_l = \text{transformer\_block}(a_{l-1})$$

$$P(u) = \text{softmax}(a_l W_e^T)$$

Where W_e is the word embedding matrix and W_p is the position embedding matrix.

For the supervised part, we have labeled data with value $X=[x^{<1>},x^{<2>},\ldots,x^{<m>}]$ and label y. The input X into the pre-trained decoder to obtain transformer block's activation $a_l^{<m>}$, which is fed into linear output layer with parameter W_y, to predict y:

$$P(y|x^{<1>},x^{<2>},....,x^{<m>}) = \text{softmax}(a_l^{<m>} W_y)$$

Which gives objective function:

$$L_2(X) = \sum_{(x,y)} logP(y|x^{<1>},.....,x^{<m>})$$

Then, the auxiliary objective is constructed as:

$$L_3(X) = L_2(X) + \lambda * L_1(X)$$ [25].

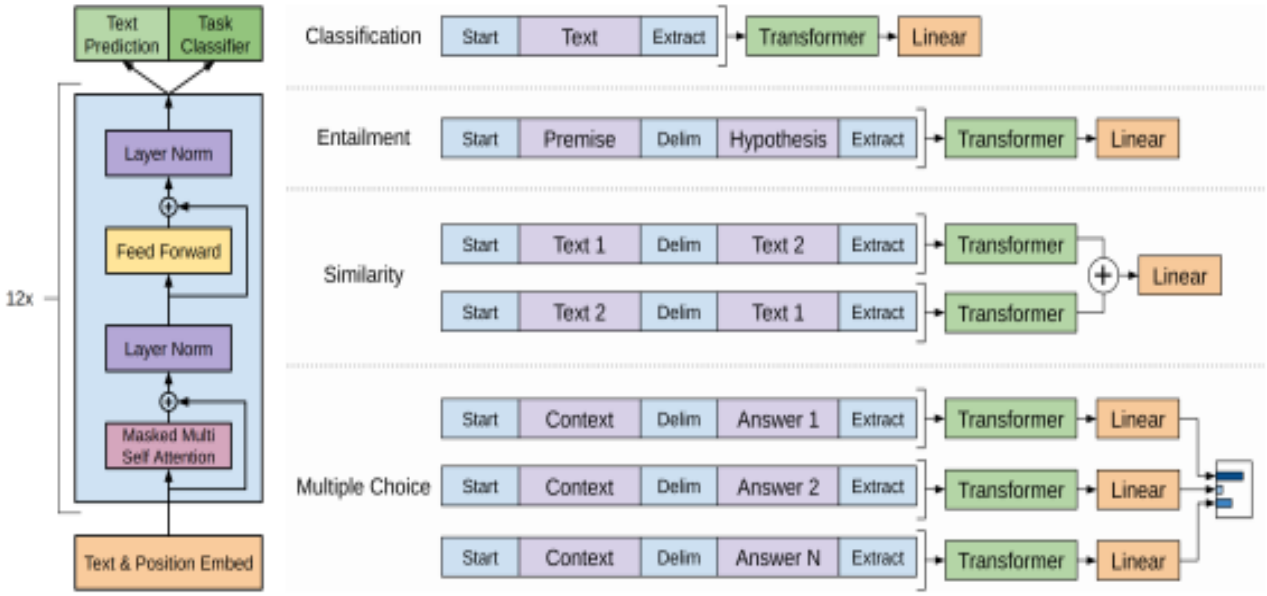


Figure 8: (left) Transformer architecture and training objectives used in this work. (right) Input transformations for fine-tuning on different tasks [25]

With the success of GPT on a wide range of NLP tasks, outperforming even the fine-tuned task-specific models, the upcoming path for the next models was clear; larger dataset, larger models. The paper on GPT-2 "Language Models are Unsupervised Multi Task Learners" [26] went along that path and introduced a model larger than GPT with a larger dataset scraped from the internet. Researchers scraped the posts from the social media platform, Reddit, with at least 3 karmas. The resulting dataset, WebText, contained over 8 million documents with 40GB of text. The model is similar to GPT but larger with 1.5B parameters, 50,257 tokens, and a context window of 1024 tokens; roughly 10X the parameter and 10X the data of GPT. Layer normalization was moved to the input of each sub-block and an additional layer normalization was added after the final self-attention[27]. Tasked Conditioning was added where the information about the task to be performed is added to the objective as P(output|input, task) instead of the regular objective P(output|input). This allows for the model to perform well on zero-shot task transfer, where the model performs tasks that are not in the training examples, and the model understands the task based on given information. The model achieved a state of the results in 7 out of 8 tasks in a zero-shot setting.GPT-2 is able to translate texts, summarizes passages, answer questions and generate outputs on that level which is sometimes indistinguishable from humans[28].

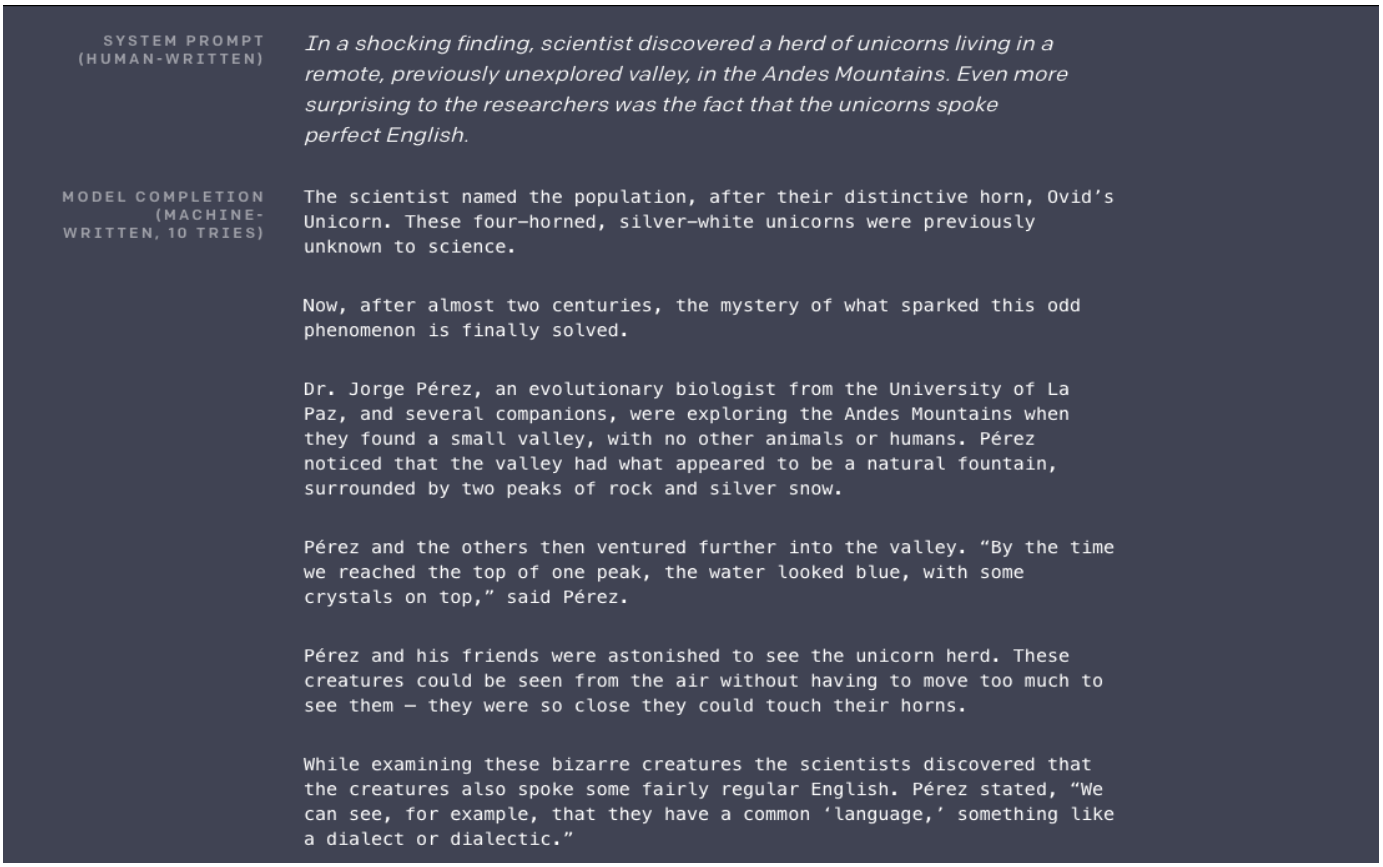


Fig 9: A text generation sample from OpenAI's GPT-2 language model[29]

The next iteration of GPT-2 was GPT-3 which is trained on a massive 175 billion parameter making it 100X the size of its predecessor, GPT-2[29]. The goal was to reduce supervised fine-tuning i.e improving performance on zero-shot transfer. The paper on GPT-3 discussed in-context learning, where the model learns a wide range of patterns and uses it during inference time to recognize the given text. The model was similar to GPT-2 with few modifications. The size of word embedding was increased to 12888 from 1600 in GPT-2, the context window size was increased to 2048 tokens and alternating dense and locally banded sparse attention patterns were used. The training dataset consisted of 5 datasets namely: Common Crawl, WebText2, Books1, Books2, and English Wikipedia. GPT-3 is stronger than GPT-2 and is capable of handling more niche topics. Due to the concern of possible misuse, the model can only be accessed through API, whereas Microsoft holds the exclusive license to the underlying workings of GPT-3. Compared to GPT-2, GPT-3 can now go further with tasks such as answering questions, writing essays, text summarization, language translation, and the ability for it to be able to generate computer code is already a major feat unto itself.[30].

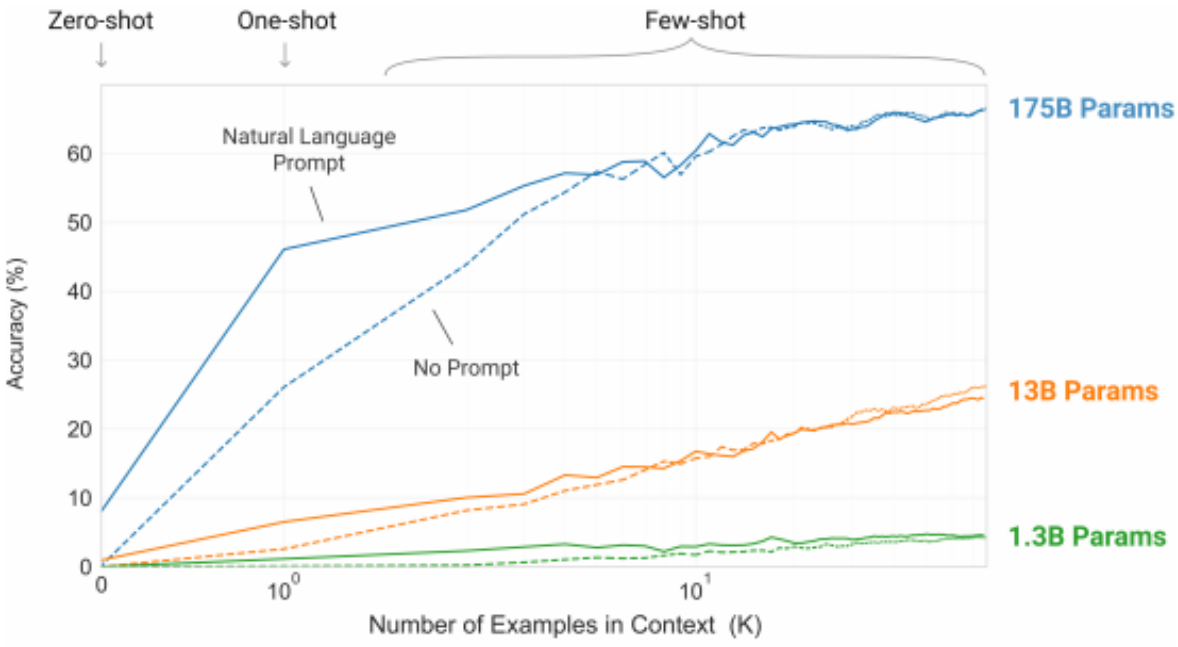


Fig 10: Comparison between GPT-1,2 and 3 on zero-shot, one-shot, and few-shot transfer tasks[30]

GPT-3 was able to generate codes in python, SQL, js, and other languages. But the accuracy was near zero. To improve on this task OpenAI created a model Codex [31] , based on GPT, specifically designed for the task of programming. On June 29, 2021, OpenAI in partnership with Microsoft announced a "virtual pair programmer", GitHub Co-pilot powered by Codex. The model was trained on 54 million software repositories amounting to 159GB of data hosted on GitHub. GitHub Copilot understands significantly more context than most code assistants. So, whether it's in a docstring, comment, function name, or the code itself, GitHub Copilot uses the context you've provided and synthesizes code to match.OpenAI Codex has a broad knowledge of how people use code and is significantly more capable than GPT-3 in code generation, in part, because it was trained on a data set that includes a much larger concentration of public source code[32]. GitHub Copilot works with a broad set of frameworks and languages but its technical preview works especially well for Python, JavaScript, TypeScript, Ruby and Go. Just comment on the logic you want and GitHub Co-pilot writes the required code for you[33]. The models' accuracy is 43%, and it is expected to increase with the release of larger models trained on larger datasets. This has raised a question about the relevance of programming jobs in the future. In the following section, we try to extrapolate the future of AI-assisted programming, or AI programming, based on the progress being done as of the present day.

collaborators.ts | get_repositories.py | non_alt_images.js

```
/**
 * json schema:
 * [
 *   { name: "John Doe",
 *     collaborators: ["Jane Doe", "Herbert Frapp", "Elsie McEwan"]
 *   },
 *   { name: "Jane Doe",
 *     collaborators: ["John Doe", "Karen Smith"]
 *   },
 *   { name: "Skittles the Cat",
 *     collaborators: []
 *   }
 * ]
 */
function collaborators_map(json: any): Map<string, Set<string>> {
  const map = new Map<string, Set<string>>();
  for (const item of json) {
    const name = item.name;
    const collaborators = item.collaborators;
    const set = new Set<string>(collaborators);
    map.set(name, set);
  }
  return map;
}
```

Copilot

Fig 11: GitHub Copilot[33]

III. FUTURE OF PROGRAMMING AS IT PERTAINS TO THE GROWTH OF AI.

The current developments in NLP[25][26][30][31][34] have proved that larger models trained on larger datasets have higher accuracy. With a plethora of data available on the internet and growing computation power, the size and accuracy of NLP have been growing with time as depicted in fig 10 and it can be predicted with high confidence that it will further increase given the rate of current growth. Along with it, the accuracy of AI-powered programming assistants like

GitHub copilot is likely to increase at a similar rate. This has raised questions about the need for human programmers in the future.

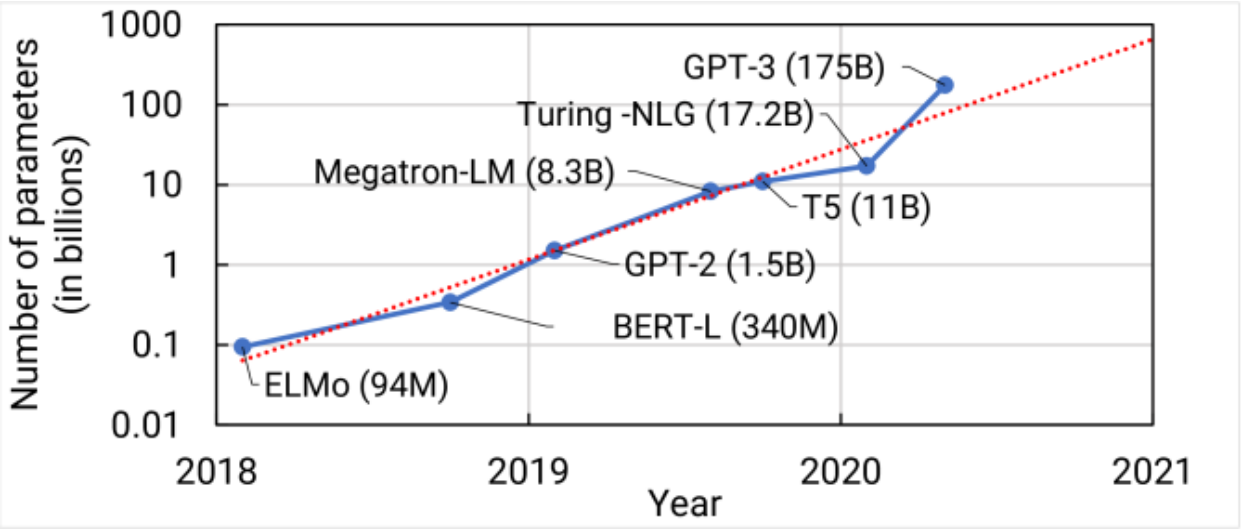


Fig 12: Trend of sizes of state-of-the-art Natural Language Processing (NLP) models with time [35]

Many AI ML and NLP applications have emerged to reduce software development complexities like errors and defects and to automate testing and development time, such as Bayou (2018), Sketch2Code (2018), DeepCode (2016), Embold (2010), PyLint, CleverCommit (2019), GitHub Copilot (2021), and so on. All these applications aid the experience of a programmer from code review to auto-completion and from debugging to vulnerability testing.

With the release of GitHub copilot, the focus has also been shifted to code assistant models, instead of just the general-purpose models. Shortly after the release of the GitHub Co-pilot, OpenAI showcased an improved version of Codex which powered the co-pilot. They demonstrated making a game through commands in normal human language. Eg: While given the prompt "When the rocket is clicked, temporarily display some text saying Fire thrusters…..and temporarily speed up to 4x for 0.25 second", the codex automatically generated code to add these features to the rocket on the screen.

So we can see that programming and software development is very different now than it was 20 or 30 years ago. The automation of AI in programming is used to develop better quality economics software and improve the user experience. Today, AI products and services demand a lot of real-time embedded software for stand-alone IoT devices. So programmers have to brace this disruptive technology and they have to be adaptive to learn the new technology as they evolve.

Although these models generating codes upon simple prompts seems impressive, it is far from being useful for commercial purposes. The accuracy of Codex is 37%, which is a huge improvement over GPT-3's 0%, but the accuracy is not good enough. Models dedicated to the coding task have just started to take off. Even while there are currently

artificial intelligence tools that can create rudimentary code, they have no method to identify which features to prioritize or what problem a piece of software in development would answer.
The only person who can write code that is based on knowledge of exact specifications and needs is a brilliant programmer. In addition, as the example illustrates, only programmers are able to make sense of complex problems that don't have specific solutions or numerous alternative responses.
Programming is not all about coding, it involves design and engineering systems. So, coding models have a long journey ahead. Along with coding, the models will also have to be proficient in designing efficient systems.

The current research[32] and demonstrations have been showing models making programs from human given prompts. Most of the time, the prompts have to include technical details, which definitely requires a human engineer or coder. Furthermore, the model is trained on public repositories on GitHub. Certainly, all of those repositories do not contain industry-standard codes. So, the code generated by the model is often not efficient. A programmer may use it to generate some code and improve on it to use on his software. So, as of the present condition, coding assistants are a helpful tool that might save programmers some time.

A Twitter bot named Tay was released in 2016 by Microsoft. A 19-year-old American girl's linguistic patterns were mimicked, and she learned through interacting with human Twitter users. Just 16 hours after its introduction, Microsoft had to take Tay offline due to its abusive tweeting.
However, this isn't the only AI problem on record. Facebook's bots, Bob and Alice, had to be shut down at the beginning of 2017. They were designed to facilitate communication between humans and computers. It wasn't until the bots were instructed to speak with each other that they began to converse in a way that was hard for people to understand.

If the input data are incorrect or polluted, an AI model is said to provide biased results. Reference [36] depicts in Figure 13 the risk level of AI-integrated applications. In Figure 13, the x-axis shows the level of automation and the y-axis shows the point of application of AI. The level of automation is on a scale of 1 to 10. Scale 1 is a low level of automation where humans have full control in deciding the next level of action and scale 10 is highly automated where the software automatically decides the next level of action and informs humans if it is needed. Point of application) of AI can be either process, product, or runtime. The risk of automation rises with increasing application points and automation levels. Accordingly, software firms should take calculated risks in accordance with their AI competence in order to leverage AI benefits.

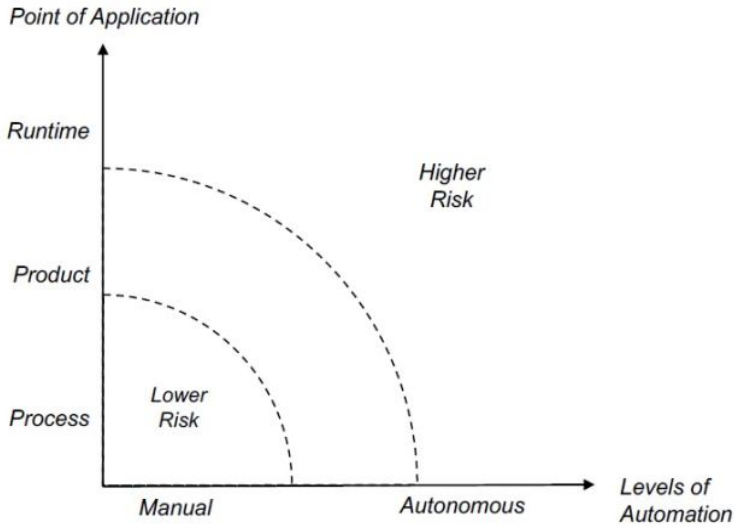


Fig 13: Risk Factor in Level of Automation and Point of Application [36]

Though we cannot predict for certain if AI will replace human programmers. One thing we can say for certain is that programmers without knowledge of AI will be replaced with programmers with knowledge of AI. AI has been becoming an integral part of programming, and it definitely will be an invaluable part of programming in the upcoming days. Programming is likely to be less explicit in upcoming days, with programmers building AI models which solve the problem, instead of programmers explicitly designing and coding the solutions based on the task. So, programmers should continuously update their skills and be knowledgeable with state-of-the-art AI models.

IV. CONCLUSION

To conclude, we can say that computers will program much better in the near future than they currently do. Although the current models are quite imprecise, they will not be the same. Better models trained on improved solutions can dramatically improve their performance over time. Finally, the day may come when programming assistants will actually be promoted to “pair programmers”. It may take years or decades, but programmers will have to learn to adapt to it. At this point, the use of models will increase efficiency and reduce time. They won't completely replace programmers as the basic design will still be done by programmers and engineers and the models will help produce well-written code. However, it is also true that the next generation of programmers should focus more on knowing what to code to build a particular feature in the best way and the logic needed than on just knowing how to code based on instructions provided as these models would be present if the task was only to code pre-existing functions and logic.

There's little doubt that Stephen Hawking recognized a serious risk in computers becoming intelligent. He did, however, give some advice: According to Hawking, "we urgently need to establish direct links to the brain so that computers may complement human intellect rather than be in opposition to it.

ACKNOWLEDGEMENT

We are immensely grateful to Mr.Andrew Ng and his team at deeplearning.ai, for providing high-quality materials on deep learning for free. Our paper is largely inspired by and based on the course Deep Learning Specialization on Coursera. We would also like to thank Mr. Ashish Neupane, Mentor: Incubate Nepal(2nd Iteration) for his valuable time and effort in reviewing our paper and providing his valuable feedback.

REFERENCES

[1] S. (2020, April 23). *The History of Artificial Intelligence*. Science in the News. Available : https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

[2] *Logic Theorist - Complete History of the Logic Theorist Program*. History Computer. Available: https://history-computer.com/inventions/logic-theorist-complete-history-of-the-logic-theorist-program/

[3] *DARPA, ARPA, Defense / Advanced Research Project Agency*. (2020, November 24). LivingInternet. Available: https://www.livinginternet.com/i/ii_darpa.htm

[4] Markoff, J. (2013, February 5). *On 'Jeopardy!' Watson Win Is All but Trivial*. The New York Times. Available: https://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html

[5] Yann LeCun (2016). Slides on Deep Learning OnlineArchived 2016-04-23 at the Wayback Machine.

[6] *Robots from the Republic of Korea and the United States take home $3.5 million in prizes*. DRC Finals. Available: https://web.archive.org/web/20150611162358/http://theroboticschallenge.org/

[7] *Open Letter on Autonomous Weapons*. (2019, March 15). Future of Life Institute. Available: https://futureoflife.org/open-letter-autonomous-weapons/

[8] *Dota 2*. (2020, September 2). OpenAI. Available : https://openai.com/blog/dota-2/

[9] *Alphago-the-story-so-far*. DeepMind. Available: https://deepmind.com/research/case-studies/alphago-the-story-so-far

[10] Khurana, Diksha & Koli, Aditya & Khatter, Kiran & Singh, Sukhdev. (2017). Natural Language Processing: State of The Art, Current Trends and Challenges. ResearchGate

[11] Nation, K., Snowling, M. J., & Clarke, P. (2007). Dissecting the relationship between language skills and learning to read: Semantic and phonological contributions to new vocabulary learning in children with poor reading comprehension. Advances in Speech
Language Pathology, 9(2), 131-139. Available: https://doi.org/10.1080/14417040601145166

[12] Introduction to Linguistics for Natural Language Processing. Ted Briscoe Computer Laboratory University of Cambridge c Ted Briscoe, Michaelmas Term 2013. Department of Science and Technology. University of Cambridge.

[13] Evaluating one-hot encoding finite state machines for SEU reliability in SRAM-based FPGAs | IEEE Conference Publication | IEEE Xplore

[14] Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781v3 [cs.CL]
[15] GloVe: Global Vectors for Word Representation, glove.pdf. Stanford University
[16] Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. arXiv:1808.03314v9 [cs.LG]
[17] CS 230 - Recurrent Neural Networks Cheatsheet. Stanford University
[18] *The Unreasonable Effectiveness of Recurrent Neural Networks*. (2015, May 21). Andrej Karpathy Blog. Available: http://karpathy.GitHub.io/2015/05/21/rnn-effectiveness/
[19] LONG SHORT-TERM MEMORY. Neural Computation 9(8):1735-1780, 1997. Available: https://www.bioinf.jku.at/publications/older/2604.pdf
[20] Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv:1412.3555v1 [cs.NE]
[21] GLoVe-LSTM | Papers With Code. arXiv:1412.3555v1 [cs.NE]
[22] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," in IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673-2681, Nov. 1997, doi: 10.1109/78.650093. (Bidirectional recurrent neural networks | IEEE Journals & Magazine | IEEE Xplore)
[23] Rahman MM, Watanobe Y, Nakamura K. A Bidirectional LSTM Language Model for Code Evaluation and Repair. *Symmetry*. 2021; 13(2):247. Available: https://doi.org/10.3390/sym13020247
[24] Attention Is All You Need. arXiv:1706.03762 [cs.CL]
[25] Improving Language Understanding by Generative Pre-Training language_understanding_paper.pdf. OpenAI
[26] Radford, A. (2021, June 22). *Better Language Models and Their Implications*. OpenAI. Available: https://openai.com/blog/better-language-models/
[27] Shree, P. (2020, November 10). *GPT models explained. Open AI's GPT-1,GPT-2,GPT-3 | Walmart Global Tech Blog*. Medium. Available: https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2
[28] Hern, A. (2019, February 15). *New AI fake text generators may be too dangerous to release, say creators*. The Guardian. Available: https://www.theguardian.com/technology/2019/feb/14/elon-musk-backed-ai-writes-convincing-news-fiction
[29] Language Models are Few-Shot Learners. arXiv:2005.14165v4 [cs.CL]
[30] *GPT-2 (GPT2) vs. GPT-3 (GPT3): The OpenAI Showdown*. Dzone.Com. Available: https://dzone.com/articles/gpt-2-gpt2-vs-gpt-3-gpt3-the-openai-showdown
[31] Evaluating Large Language Models Trained on Code. arXiv:2107.03374v2 [cs.LG]
[32] Friedman, N. (2021, June 29). *Introducing GitHub Copilot: your AI pair programmer*. The GitHub Blog. Available; https://GitHub.blog/2021-06-29-introducing-GitHub-copilot-ai-pair-programmer/
[33] *GitHub Copilot · Your AI pair programmer*. GitHub Copilot. Available: https://copilot.GitHub.com/
[34] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805v2 [cs.CL]
[35] Efficient Large-Scale Language Model Training on GPU Clusters. arXiv:2104.04473v5 [cs.CL]
[36] Robert Feldt et al. Ways of Applying Artificial Intelligence in Software Engineering. arXiv:1802.02033v2[cs.SE].

