Unit 5

AI Languages

5.1 Evolution and Historical Perspective

The evolution of artificial intelligence grew with the complexity of the languages available for development. In 1959, Arthur Samuel developed a self-learning checkers program at IBM on an IBM® 701 computer using the native instructions of the machine (quite a feat given search trees and alpha-beta pruning). But today, AI is developed using various languages, from Lisp to Python to R. The programming languages that are used to build AI and machine learning applications vary. Each application has its own constraints and requirements, and some languages are better than others in particular problem domains. Languages have also been created and have evolved based on the unique requirements of AI applications.

Early in the history of AI, the only languages that existed were the native languages of the machines themselves. These languages, called machine language or assembly language, were cumbersome to use because simple operations were the only operations that existed. For example, move a value from memory to a register, subtract the contents of a memory address from the accumulator, and other such operations. Likewise, the data types of the machine were the only types available, and they were restricted. However, even before high-level languages appeared, complex AI applications were being developed.

In 1956, one of the founding fathers of AI, John McCarthy, created a tree search pruning scheme called alpha-beta pruning. This work occurred at a time when many AI problems were considered search problems and while considerable research activity was happening. Memory and compute power were also limited, but this technique allowed researchers to implement more complex problems on early computer systems with limited resources. The alpha-beta pruning technique was applied to early applications of AI in games.

Also in 1956, Arthur Samuel developed a checkers-playing program on an IBM 701 computer using McCarthy's alpha-beta search. However, Samuel's game included an advantageous element: *Rather than playing the checkers program himself to teach it how to play, Samuel introduced the idea of self-learning and allowed the program to play itself*. Samuel developed his program in the native instruction set of the IBM 701 system, which was quite a feat given the complexity of his application and the low-level instructions at his disposal.

5.1.1 Early Years (1954 – 1973)

The early years were a time of discovery—the introduction of new machines and their capabilities and the development of high-level languages that could use the power of these machines for a broad range of applications. In 1958, a chess-playing program called NSS (named after its authors, Newell, Shaw, and Simon) was developed for the IBM 704 computer. This program viewed chess in terms of search and was developed in Information Processing Language (IPL), also developed by the authors of NSS.

IPL was the first language to be developed for the purpose of creating AI applications. IPL was a higher-level language than machine language, but only slightly. It did, however, permit developers to use the language on various computer systems. IPL introduced numerous features that are still used today, such as lists, recursion, higher-order functions, symbols, and even generators that could map a list of elements to a function that would iterate and process the list. The first version of IPL was never implemented, but subsequent versions (2 - 6) were implemented and used on systems like the IBM 704, IBM 650, and IBM 7090, among others. Some of the other early AI applications that were developed in IPL include Logic Theorist and General Problem Solver.

Despite the success of IPL and its wide deployment on the computer architectures of the day, IPL was quickly replaced by an even higher-level language that is still in use almost 60 years later: LISP. IPL's esoteric syntax gave way to the simpler and more scalable LISP, but IPL's influence can be seen in its later counterpart, particularly its focus on lists as a core language feature. LISP—the LISt Processor—was created by John McCarthy in 1958. McCarthy's goal after the Dartmouth Summer Research Project on AI in 1956 was to develop a language for AI work that was focused on the IBM 704 platform. FORTRAN was introduced in 1957 on the same platform, and work at IBM extended FORTRAN to list processing in a language called the FORTRAN List Processing Language (FLPL). This language was used successfully for IBM's plane geometry project, but as an extension to FORTRAN, FLPL lacked some key features. LISP was a foundational programming language and implemented many of the core ideas in computer science, such as garbage collection, trees, dynamic typing, recursion, and higher-order functions. LISP not only represented data as lists but even defined the source code itself as lists. This feature made it possible for LISP to manipulate data as well as LISP code itself. LISP is also extensible, allowing programmers to create new syntax or even new languages (called domain-specific languages) to be embedded within LISP.

In 1968, Terry Winograd developed a ground-breaking program in LISP called SHRDLU that could interact with a user in natural language. The program represented a block world, and the user could interact with that world, directing the program to query and interact with the world using statements such as "pick up the red block" or "can a pyramid be supported by a block?" This demonstration of natural language understanding and planning within a simple physics-based block world created considerable optimism for AI and the LISP language.

5.1.2 Turbulent Years (1974 – 1993)

The turbulent times represent a period of instability in the development and funding of AI applications. This era began with the first AI winter where funding disappeared because of a failure to meet expected results. In 1980, expert systems rekindled excitement for and funding of AI, but by 1987, the AI bubble burst again, despite the advancements made during this time, which led to the second AI winter.

LISP continued to be used in a range of applications during this time and proliferated through various dialects. LISP lived on through Common LISP, Scheme, Clojure, and Racket. The ideas behind LISP continued to advance through these languages and others outside the functional domain. LISP continues to power the oldest computer algebra system, called Macsyma (Project

MAC's SYmbolic MAnipulator). Developed at the Massachusetts Institute of Technology's AI group, this computer algebra environment is the grandfather of many programs, like Mathematica, Maple, and many others. Other languages began to appear in this time frame, not necessarily focused on AI but fueling its development. The C language was designed as a systems language for UNIX systems but quickly grew to one of the most popular languages (with its variants, such as C++), from systems to embedded device development.

A key language in this time was developed in France and called Prolog (Programming in Logic). This language implemented a subset of logic called Horn clauses and allowed information to be represented by facts and rules and to allow queries to be executed over these relations. The following simple Prolog example illustrates the definition of a fact (Socrates is a man) and a rule that defines that if someone is a man, he is also mortal. Prolog continues to find use in various areas and has many variants that incorporate features such as object orientation, the ability to compile to native machine code, and interfaces to popular languages (such as C). One of the key applications of Prolog in this time frame was in the development of expert systems (also called production systems). These systems supported the codification of knowledge into facts, and then rules used to reason over this information. The problem with these systems is that they tended to be brittle and maintaining the knowledge within the system was cumbersome and error prone.

An example expert system was the eXpert CONfigurer (XCON), which was used to configure computing systems. XCON was developed in 1978 in OPS5 (a production system language written in LISP) that used forward-chaining for inference. By 1980, XCON was made up of 2,500 rules but was too expensive to maintain. Prolog and LISP were not the only languages used to develop production systems. In 1985, the C Language Integrated Production System (CLIPS) was developed and is the most widely used system to build expert systems. CLIPS operates on a knowledge system of rules and facts but is written in C and provides an interface to C extensions for performance.

The failure of expert systems was one factor that led to the second AI winter. Their promise and lack of delivery resulted in significant reductions in funding for AI research. However, new approaches rose from this winter, such as a revival of connectionist approaches, bringing us to the modern era.

5.1.3 Modern Era (1994 – Present)

The modern era of AI brought a practical perspective to the field and clear success in the application of AI methods to real-world problems, including some problems from early in AI's history. The languages of AI also showed an interesting trend. While new languages were applied to AI problems, the workhorses of AI (LISP and Prolog) continued to find application and success. This era also saw the revival of connectionism and new approaches to neural networks, such as deep learning.

The explosion of LISP dialects resulted in a unification of LISP into a new language called Common LISP, which had commonality with the popular dialects of the time. In 1994, Common LISP was ratified as American National Standards Institute Standard X3.226-1994. Diverse programming languages began to appear in this time frame, some based on new ideas in computer science, others focused on key characteristics (such as multiparadigm and being easy to learn). One key language fitting this latter category is Python. Python is a general-purpose interpreted language that includes features from many languages (such as object-oriented features and functional features inspired by LISP). What makes Python useful in the development of intelligent applications is the many modules available outside the language. These modules cover machine learning (scikit-learn, Numpy), natural language and text processing (NLTK), and many neural network libraries that cover a broad range of topologies.

The R language (and the software environment in which you use it) follows the Python model. R is an open-source environment for statistical programming and data mining, developed in the C language. Because a considerable amount of modern machine learning is statistical in nature, R is a useful language that has grown in popularity since its stable release in 2000. R includes a large set of libraries that cover various techniques; it also includes the ability to extend the language with new features.

The C language has continued to be relevant in this time. In 1996, IBM developed the smartest and fastest chess-playing program in the world, called Deep Blue. Deep Blue ran on a 32-node IBM RS/6000 computer running the IBM AIX® operating system and was written in C. Deep Blue could evaluate 200 million positions per second. In 1997, Deep Blue became the first chess AI to defeat a chess grandmaster. IBM returned to games later in this period, but this time less structured than chess. The IBM Watson® question-and-answer system (called DeepQA) was able to answer questions posed in natural language. The IBM Watson knowledge base was filled with 200 million pages of information, including the entire Wikipedia website. To parse the questions into a form that IBM Watson could understand, the IBM team used Prolog to parse natural-language questions into new facts that could be used in the IBM Watson pipeline. In 2011, the system competed in the game Jeopardy! and defeated former winners of the game.

With a return to connectionist architectures, new applications have appeared to change the landscape of image and video processing and recognition. Deep learning (which extends neural networks into deep, layered architectures) are used to recognize objects in images or video, provide textual descriptions of images or video with natural language, and even pave the way for self-driving vehicles through road and object detection in real time. These deep learning networks tend to be so large that traditional computing architectures cannot efficiently process them. However, with the introduction of graphics processing units (GPUs), these networks can now be applied. To use GPUs as neural network accelerators, new languages were needed to bring traditional CPUs and GPUs together. An open standard language called the Open Computing Language (OpenCL) allows C- or C++-like programs to be executed on GPUs (which consist of thousands of processing elements, simpler than traditional CPUs). OpenCL allows parallelism of operations within GPUs orchestrated by CPUs.

5.2 Successes of AI

Optimization of AI language has enabled a revolutionary change across many sectors globally. For instance, AI has been deployed across different fields like finance, health, engineering and education and the proliferation of AI languages across these sectors has helped birth new tech businesses. In Education and Research, AI has brought about significant improvement in the quality of delivery of educational resources across the globe, expert systems have been created to provide seamless learning content to students and researchers across borders. In the health sector, AI has been used to assist in automated data management, development of artificial neural network to assist in rapid patient care. The advancements in AI have provided further dynamics of analysis of face or object recognition techniques for audiovisual. Also, in music evolution in composing human like notes. The invasion of cognitive problem-solving skill has brought innovative ideas that is answering engineering questions.

5.3 Challenges and Limitations of AI

The main aim of artificial intelligence is to build an intelligent machine that will make life easier for human beings. The machine should be able to think like humans and some intelligence traits added to it. The programmers want to build some emotional quotient into the machines. Expert development is one of the major problems of AI programming language. LISP, a functional language, created as a mathematical notation for computer programmer was developed for lambda calculus which is not part of undergraduate curriculum in higher institutions. This makes it more tedious for beginners to master LISP compared to other object -oriented language like JAVA. The expert community and library capacity are limited due to this difficulty. Since AI involves building a machine that is intelligent like human beings, it must also face some challenges like humans. Identifying some of these challenges will minimize the associated risks and at the same time make sure that we take full advantage of this technology.

Most researchers believe and agree that a super intelligence AI is unlikely to showcase human emotions like love or hate and that therefore, it cannot become intentionally benevolent or malevolent. However, the most likely scenario where it can pose a threat to the society is via autonomous weapons. These are weapons that AI systems are programmed to use to kill. If in the hands of the wrong person, these weapons could easily cause mass casualties. This could even lead to an AI war that would also result in mass causalities. Legal challenges related to AI's application in the financial industry could be related to the consequences of erroneous algorithms and data governance. Erroneous algorithms, due to the lack of appropriate data, can leave a big dent in the profits of an organization by making incorrect and perhaps detrimental predictions. Poor data governance can result in data breaches where customers' PII (personal identifiable information) that acts as a feedstock to an algorithm may get into the hands of hackers and can cause legal challenges for the organization.

5.4 AI Languages

Language	**Capabilities/ Features**	**Limitations**	**Applications**
LISP	1. Lisp is a mathematical language that uses Symbolic data processing (S-expressions). These S-expressions are stored in a structured list. 2. It performs computational analysis on sequential programs, it has simple internal structure and compatible with other systems.	1. Ultra-slow numerical computation and lack of better representation of block of registers. 2. It has a higher overhead when compared with other conventional programming	1. Used in differential and integral calculus, electrical circuit theory, mathematical logic, game playing and other fields of artificial intelligence. 2. A vastly useful tool for the programming of many Expert Systems.

	3. Able to rapidly analyze large trees by efficiently utilizing a list-based tree structure, search space and rule-based. 4. More powerful for providing symbolic computing compared with conventional programming methods. 5. Very flexible programming language compared with most others because it is built around a kernel of mathematical principles. 6. Lisp has potentials for systems programming and is suitable for writing operating systems. It also possesses capabilities for heterogeneous parallel computing. 7. Coding is fast and efficient due compilers Automatic garbage collections were invented for lisp language.	languages. This has narrowed its use in AI. 3. It is limited with respect to processing and memory requirements. Its size, unwieldiness, "kitchen sink" design strategy, and general ADAfication is disliked by most of its critics. 4. Slower compared to C++.	3. It is programming language of choice of most AI researchers. 4. It has been applied in Musical composition and processing. 5.Foundational language for all programming languages.
PROLOG	1. Performs efficient implementation by incorporating parallel mode of computation. 2. Support for relational databases, natural language processing and automated reasoning. 3. It supports symbolic programming, therefore, by manipulating trigonometric relations and identities, it can derive useful kinematic equations of open kinematic chains. 4.	1. The coding of the knowledge and the organization and modularization requires a lot of creativity. Some examples include fact/consequent representation problem and existential quantification problems. 2. One of its drawbacks is the lack of natural mechanism to tackle the issue of uncertainties since by default it is designed to be a two-valued logic programming language. 3. Prolog by default provides limited support for real-life knowledge engineering. 4. To improve performance in Prolog applications, the performance of the microarchitecture of the uniprocessor engine needs to be developed. 5. Convenient logic program properties no longer hold, and upsetting practical problems sprout, namely the debugging of programs with side-effects is harder. 6. By default Prolog does not consider the dynamic nature of agents such as knowledge acquisition and action execution. This poses a problem because the agent	1. Suitably adapts to applications requiring implementation in logic programming environments. 2. Relational Database applications, natural language processing, theorem proving, automated reasoning. 3. Used for establishing expert systems in specific research activities. 4. Used in logic-circuit model building, expert systems, AI and natural-language interfacing. 5. Used in clinics for detection and classification of QRSs in Electrocardiography (ECG).

		might work in a dynamic environment where unexpected things can happen. 7. Slower compared to C++.	
LOOP (Logic and object-oriented programming language)	LOOP extends Prolog logic programming paradigm with object-oriented features.	It lacks mechanisms for structuring knowledge (program clauses).	Used in LP (Linear Programming)-based AI applications.
Archlog	Can produce high-performance designs without detailed knowledge of hardware development and a framework for designing multiprocessor architectures.		Application in machine learning and cognitive robotics.
EOLC (Epistemic Ontology Language with Constraints)	Used for specifying the epistemic ontology for heterogeneous verification.		Application in redundant flight guidance system and in the heterogeneous Verification of Embedded Control Systems.
Python	1. Python has improved over a short period of time compared to Java and C++. 2. The efficiency of the programmer is highly improved because of its support for object-oriented design, functional and procedural styles of programming. 3. It has high level syntax Algorithm can be tested without implementation.	1. It is not good for mobile computing because of its weak Language for mobile computing 2. The execution is slow in artificial intelligence development since it works with the help of an interpreter unlike C++ and Java.	
C++	1. C++ can organize data because it is a multi-paradigm programming that supports object-oriented principles. 2. It has high level of abstraction which makes it good for solving complex problem in AI.	1. C++ can organize data because it is a multi-paradigm programming that supports object-oriented principles. 2. It has high level of abstraction which makes it good for solving complex problem in AI.	1. Used in virtual robot simulation and synthesis, grasp synthesis, 3D drawing. 2. Any kind of data can be modelled and simulated easily in AI.
JAVA	Java is portable, implementation on different platforms easy because of virtual machine technology, algorithms coding is very easy.	Java's response time is more and has less execution speed, and thus, makes it slower than C++.	It enables the Automatic Speech Recognition (ASR) systems, improved performance when linked with prolog.
ADA	High performance and maintainability, list processing facility.		Development of large-scale AI software.

Other languages are AIML, C#, Smalltalk, STRIPS, POP-11, R, Haskell, Julia.

5.5 Modern AI Libraries

Sr. No.	AI Library	Release Date	Innovator	Programming Language Platform
1.	ALICE	1995	Joseph Weizenbaum	Java

2.	OpenNN	2003	International Center for Numerical Methods in Engineering (CIMNE)	C++
3.	OpenCog	2008	OpenCog Foundation	C++, Python
4.	TensorFlow	2015	Google Brain Team	Python, C++, CUDA
5.	Siri	2011	Apple	Objective C
6.	Neural Designer	Unknown	Arteinics	C++
7.	Keras	2015	François Chollet	Python
8.	Scikit-learn	2007	David Cournapeau	Python, Cython, C, C++
9.	Pandas	2017	Wes McKinney	Python
10.	SciPy	2017	Travis Oliphant, Pearu Peterson, Eric Jones	Python, Fortran, C, C++

5.6 Characteristics of AI Languages

One of the most desirable qualities of AI programming languages is *speed*. PROLOG and LISP are slower compared to C/C++ and Python. An additional ability to *support object-oriented principles* is desirable. Sometimes it may be necessary to trade between quick response time and execution speed. Other desirable qualities are *portability* and *ease of coding*. The combination of C/C++ with Python in developing most of the trending AI libraries (such as TensorFlow and Keras) may be born out of the need to complement the weakness of one language with the strength of the other and vice-versa.