PARADIGMAS PARA LA ENSEÑANZA DE INTELIGENCIA ARTIFICIAL EN LA EDUCACIÓN ECUATORIANA

PARADIGMS FOR TEACHING ARTIFICIAL INTELLIGENCE IN ECUADORIAN EDUCATION

Diego Javier Bastidas Logroño[1*]

[1] Instituto Superior Tecnológico Quito. ITQ. ORCID: https://orcid.org/0000-0003-3924-7468. Correo: diego.bastidas@itq.edu.ec

Jenny Mariuxi Zambrano Socola[2]

[2] Distrito 22D01 Joya de los Sachas - Educación. ORCID: https://orcid.org/0009-0008-8502-8673. Correo: jenny.zambrano@educacion.gob.ec

Aida Cecibel Coronel Iñaguazo[3]

[3] Distrito 22D01 Joya de los Sachas - Educación. ORCID: https://orcid.org/0009-0007-6108-9060. Correo: aida.coronel@educacion.gob.ec

Miriam Magali Ramirez Requelme[4]

[4] Distrito 22D01 Joya de los Sachas - Educación. ORCID: https://orcid.org/0009-0002-0521-2750. Correo: miriamr.ramirez@educacion.gob.ec

*** Autor para correspondencia:** diego.bastidas@itq.edu.ec

Resumen

La enseñanza de la inteligencia artificial en el ámbito educativo ha sido una experiencia única y enriquecedora, dado el papel histórico y continuo. El introducir a los estudiantes en la historia de la inteligencia artificial ha desempeñado un papel fundamental en su desarrollo. Explorar los conceptos clave de la inteligencia artificial como el sistema de producción, el razonamiento simbólico y el aprendizaje automático nos ha permitido desarrollar este tipo de paradigma como un enfoque nuevo en la educación de esta generación. El enseñar los fundamentos de la programación simbólica como manipulación de listas, definición de funciones recursivas, representación simbólica de conocimiento y manipulación de símbolos, cómo diseñar y desarrollar sistemas expertos utilizando inteligencia artificial. Esto implica enseñarles a

representar conocimiento de dominio, implementar reglas de inferencia y crear sistemas de retroalimentación para mejorar el rendimiento del sistema. El haber explorado las técnicas de aprendizaje automático simbólico, como la inducción de árboles de decisión, la inferencia bayesiana y la lógica difusa en la colaboración en proyectos de investigación. Al haberse integrado estos paradigmas, los estudiantes pueden obtener una comprensión profunda de la inteligencia artificial mientras desarrollan habilidades prácticas en programación.

Palabras clave: inteligencia artificial; enseñanza; aprendizaje; paradigmas para la enseñanza.

Abstract

Having taught artificial intelligence in the educational field has been a unique and enriching experience, given its historical and continuous role. Introducing students to the history of artificial intelligence has played a critical role in its development. Having explored the key concepts of artificial intelligence such as the production system, symbolic reasoning and machine learning has allowed us to develop this type of paradigm as a new approach in the education of this generation. Teaching the fundamentals of symbolic programming such as list manipulation, definition of recursive functions, symbolic representation of knowledge and manipulation of symbols, how to design and develop expert systems using artificial intelligence. This involves teaching them to represent domain knowledge, implement inference rules, and create feedback systems to improve system performance. Having explored symbolic machine learning techniques, such as decision tree induction, Bayesian inference and fuzzy logic in collaboration on research projects. Having integrated these paradigms, students can gain a deep understanding of artificial intelligence while developing practical programming skills.

Keywords: *inteligencia artificial; enseñanza; aprendizaje; paradigmas para la enseñanza.*

Fecha de recibido: 21/03/2024
Fecha de aceptado: 15/05/2024
Fecha de publicado: 16/05/2024

Introducción

En esta década se ha alcanzado un avance significativo en el desarrollo de la inteligencia artificial en especial en el área de sistemas expertos (McCarthy, 1960), por tal razón que los estudiantes ecuatorianos tengan un enfoque en el estudio de los mismos desde temprana edad ayudarían a salir al país del sub desarrollo (Walrond, 2009). Hay varias razones para que el estudio de la inteligencia artificial por medio del desarrollo de sistemas expertos en lugar de sistemas convencionales entre las cuales están:

- Muchos de los problemas importantes no tienen una solución algorítmica, dado que tienen origen en complejos contextos sociales o físicos. (Keene, 1989).

- Es pragmático, los seres humanos alcanzan su desarrollo en excelencia porque son seres inteligentes, por ende, si las computadoras lo adquieren y lo usan, también tendrán un grado en la ejecución de dicha labor.
- Reconoce su valor intrínseco ya que el conocimiento es una fuente escasa cuyo refinamiento y reproducción crean riqueza, hacerlo mediante la enseñanza de la elaboración sistemas expertos desde temprana edad a los humanos es una inversión a largo plazo. (Moyne, 2023)

Para ésta interacción se eligió el programa Lisp (List Processing) que es el segundo lenguaje de programación más antiguo luego de fortran, fue creado a finales de 1950 por John McCarty, que está concebido para el tratamiento de lenguaje simbólico ya que entre sus elementos estructurales más importantes son las listas y los símbolos ya que se ha convertido en el lenguaje preferido para toda persona que desee aprender e interactuar con inteligencia artificial (Abelson, 1996). Se obtiene un beneficio interesante del uso de un lenguaje tan potente como un dialecto de Lisp como su lenguaje primario de extensión: puede implementar otros lenguajes traduciéndolos a su lenguaje primario. Si su lenguaje primario es Lisp, no es tan difícil implementar otros lenguajes traduciéndolos (Seibel, 2005). Entonces si cada aplicación extensible admitía Scheme, usted podría escribir una implementación de TCL o Python o Perl en Scheme que tradujera los programas en esos lenguajes a Scheme. Entonces, usted podría cargar su traductor en cualquier aplicación y extender dicha aplicación usando su lenguaje favorito. Dado que Lisp es un lenguaje de programación completo, es posible implementar intérpretes o compiladores de otros lenguajes, incluido el pseudocódigo o lenguajes de especificación, en Lisp (Touretzky, 1989). Esto podría incluir implementaciones de sistemas de procesamiento de lenguaje natural, además el uso de Lisp podría facilitar el desarrollo, investigación o integración de modelos de lenguaje como ChatGPT en diversos contextos (Arco, 2023). Las herramientas de desarrollo de sistemas expertos más potentes del mercado están escritas en Lisp y lo utilizan como lenguaje de programación (Shapiro, 1991).

Materiales y métodos

El formato mediante el cual debe aprender el alumno acerca de la base de la inteligencia artificial en Lisp en su sintaxis se ajusta a una serie de criterios y signos convencionales (Lamkins, 2004).

Existen numerosas implementaciones de Lisp que podemos usar en cualquier sistema operativo como son:

Lisp	URL	Licencia
LispWorks	http://www.lispworks.com	comercial
Steel Bank Common Lisp	http://www.sbcl.org	libre
Allegro Common Lisp	https://franz.com	comercial
Clozure Common Lisp	https://ccl.clozure.com	libre
Clojure	https://clojure.org	libre
Racket	http://racket-lang.org	libre

Figura 1. Implementaciones Lisp.

Por ejemplo se puede tomar una de las distribuciones para windows.

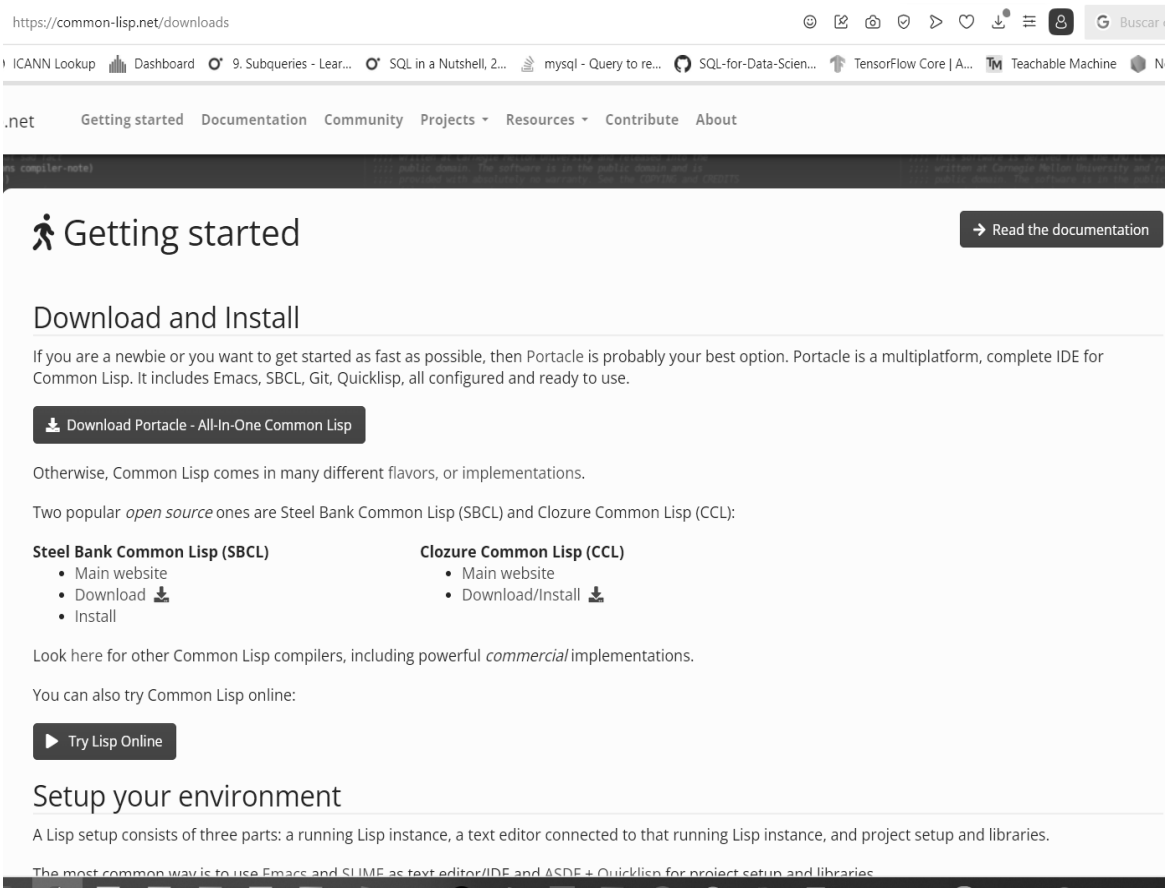

Figura 2. Instalación en Windows.

Resultados y discusión

Si bien la biblioteca estándar de funciones, tipos de datos y macros que viene con Common Lisp es bastante grande, solo proporciona construcciones de programación de propósito general. Tareas especializadas como escribir GUI, comunicarse con bases de datos y analizar XML requieren bibliotecas más allá de las que proporciona el lenguaje estandarizado ANSI (Jr., 1990). La forma más sencilla de obtener una biblioteca para hacer algo que necesita puede ser simplemente comprobar su implementación Lisp (McCarthy, Programmer´s manual, 1985). La mayoría de las implementaciones proporcionan al menos algunas funciones no especificadas en el estándar del lenguaje. Los proveedores comerciales de Common Lisp tienden a trabajar

especialmente duro para proporcionar bibliotecas adicionales para su implementación con el fin de justificar sus precios (Watson, 2024). Allegro Common Lisp, Enterprise Edition de Franz, por ejemplo, viene con bibliotecas para analizar XML, hablar SOAP, generar HTML, conectarse a bases de datos relacionales y construir interfaces gráficas de varias maneras, entre otras cosas. LispWorks, otro Lisp comercial destacado, proporciona varias bibliotecas similares, incluido un kit de herramientas GUI portátil de gran prestigio, CAPI, que se puede utilizar para desarrollar aplicaciones GUI que se ejecutarán en cualquier sistema operativo en el que se ejecute LispWorks.

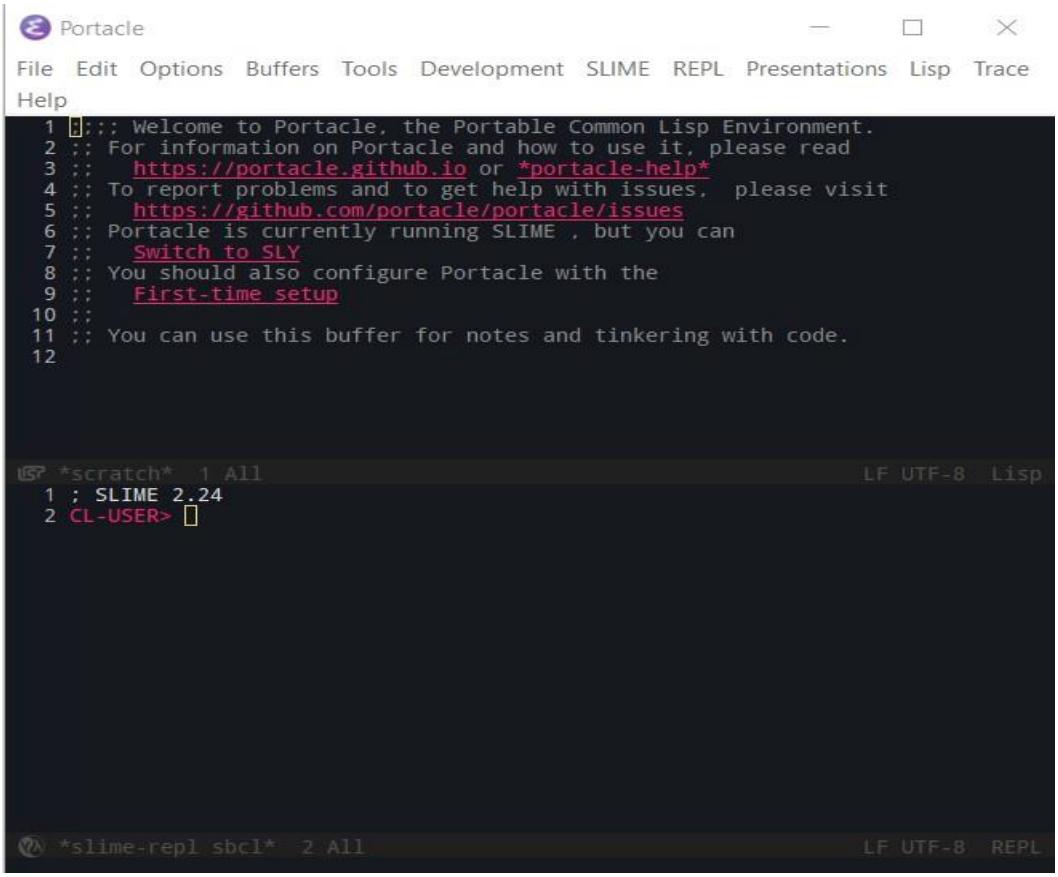

Figura 3. Portacle.

Las implementaciones gratuitas y de código abierto de Common Lisp normalmente no incluyen tantas bibliotecas empaquetadas, sino que dependen de bibliotecas portátiles gratuitas y de código abierto. Pero incluso esas implementaciones suelen cubrir algunas de las áreas más importantes que no aborda el estándar del lenguaje, como las redes y los subprocesos múltiples. La única desventaja de utilizar bibliotecas específicas de implementación es que lo vinculan a la implementación que las proporciona. Si entrega aplicaciones para el usuario final o implementa una aplicación basada en servidor en un servidor que usted controla, puede que eso no importe mucho. Pero si quieres escribir código para compartir con otros Lispers o

si simplemente no quieres estar atado a una implementación particular, es un poco más molesto. Para las bibliotecas portátiles (portátiles ya sea porque están escritas completamente en Common Lisp estándar o porque contienen una condicionalización de tiempo de lectura apropiada para trabajar en múltiples implementaciones1), lo mejor que puede hacer es ir a la Web. Con las advertencias habituales sobre las URL que se vuelven obsoletas tan pronto como se imprimen en papel, estos son tres de los mejores puntos de partida actuales:

The Common Lisp Open Code Collection (CLOCC) (http://clocc.sourceforge.net/) es una colección un poco más antigua de bibliotecas de software libre, que están destinadas a ser portátiles entre implementaciones de Common Lisp y autónomas, sin depender de bibliotecas no incluidas en el propio CLOCC.

Cliki (http://www.cliki.net/) es una wiki dedicada al software libre en Common Lisp. Si bien, como cualquier wiki, puede cambiar en cualquier momento, normalmente tiene bastantes enlaces a bibliotecas, así como a varias implementaciones de Common Lisp de código abierto. El software del mismo nombre que ejecuta también está escrito en Common Lisp.

Si bien muchas bibliotecas útiles se pueden escribir en Common Lisp "puro" utilizando sólo las características especificadas en el estándar del lenguaje, y muchas más se pueden escribir en Lisp utilizando funciones no estándar proporcionadas por una implementación determinada, en ocasiones es más sencillo utilizar una biblioteca existente escrita en otro idioma, como C. El estándar del lenguaje no especifica un mecanismo para que el código Lisp llame al código escrito en otro lenguaje ni siquiera requiere que las implementaciones proporcionen dicho mecanismo. Pero hoy en día, casi todas las implementaciones de Common Lisp soportan lo que se llama una Interfaz de Función Extranjera, o FFI para abreviar. El trabajo básico de una FFI es permitirle darle a Lisp suficiente información para poder vincular el código externo.

Como se ha dicho muchas veces, y atribuido de diversas maneras a Donald Knuth, C.A.R. Hoare y Edsger Dijkstra, la optimización prematura es la raíz de todos los males.4 Common Lisp es un lenguaje excelente para programar si desea seguir esta sabiduría pero aún necesita un alto rendimiento. Esto puede resultarle una sorpresa si ha oído la opinión generalizada de que Lisp es lento. En los primeros días de Lisp, cuando las computadoras se programaban con tarjetas perforadas, las características de alto nivel de Lisp pueden haberlo condenado a ser más lento que la competencia, es decir, el ensamblaje y FORTRAN. Pero eso fue hace mucho tiempo. Mientras tanto, Lisp se ha utilizado para todo, desde crear complejos sistemas de inteligencia artificial hasta escribir sistemas operativos, y se ha trabajado mucho para descubrir cómo compilar Lisp en un código eficiente. En esta sección hablaré sobre algunas de las razones por las que Common Lisp es un lenguaje excelente para escribir código de alto rendimiento y algunas de las técnicas para hacerlo. La primera razón por la que Lisp es un lenguaje excelente para escribir código de alto rendimiento es, irónicamente, la naturaleza dinámica de la programación Lisp, precisamente lo que originalmente hizo difícil llevar el rendimiento de Lisp a los niveles alcanzados por los compiladores FORTRAN.

La razón por la que las características dinámicas de Common Lisp facilitan la escritura de código de alto rendimiento es que el primer paso para escribir código eficiente es encontrar los algoritmos y estructuras de datos correctos. Las características dinámicas de Common Lisp mantienen el código flexible, lo que facilita probar diferentes enfoques. Dada una cantidad finita de tiempo para escribir un programa, es mucho más probable que termines con una versión de alto rendimiento si no dedicas mucho tiempo a entrar y salir de

callejones sin salida. En Common Lisp, puedes probar una idea, ver que no va a ninguna parte y seguir adelante sin haber pasado mucho tiempo convenciendo al compilador de que tu código es digno de ejecutarse y luego esperar a que termine de compilarse. Puede escribir una versión sencilla pero ineficiente de una función (un boceto de código) para determinar si su enfoque básico es sólido y luego reemplazar esa función con una implementación más compleja pero más eficiente si determina que lo es (Santos, 2019) y si el enfoque general resulta ser defectuoso, entonces no ha perdido mucho tiempo ajustando una función que ya no es necesaria, lo que significa que tiene más tiempo para encontrar un mejor enfoque.

Es imperativo el uso de lisp en ets tiempo ya que para muchos desarrolladores de software según su criterio es un lenguaje ambiguo, sin embargo este análisis en ésta investigación pretende revolucionar con la industria de ingeniería de software en lisp ya que cualquier aspecto del aprendizaje u otra característica de la inteligencia puede en principio ser descrita con precisión de tal forma que se puede construir una máquina que la simule. (McCarthy, Programmer´s manual, 1985).

Una característica importante de Lisp es su capacidad para utilizar funciones recursivas. La recursión es un concepto en programación donde una función se llama a sí misma dentro de su propia definición. Esto permite que las funciones resuelvan problemas complejos dividiéndolos en sub problemas más pequeños que se resuelven de la misma manera. En Lisp, la recursividad se expresa llamando funciones dentro de otras funciones o mediante funciones mismas. Esto hace posible construir algoritmos eficientes y elegantes para resolver una amplia gama de problemas. La recursión es una poderosa herramienta en Lisp que permite soluciones elegantes y concisas a problemas complejos y una solución óptima a un sistema experto cualquiera que sea el ámbito de la ciencia en desarrollo de software inteligente para ser resuelto con sistemas inteligentes. (Santos, 2019).

Conclusiones

- Al entender la forma en que la inteligencia artificial revoluciona actualmente por medio de chatgpt la comodidad del ser humano al usar dicha programación.
- Entender que partiendo desde lisp se pueden hacer nuevos chatgpt y mejoras significativas al avance de los sistemas expertos.
- Se puede partir de un grupo de personas dedicadas a aprender lisp para competir con los actuales programas de inteligencia artificial desde el Ecuador.

Referencias

Abelson, H. (1996). Structure and Interpretation of Computer Programs, Second Edition. Cambridge.

Arco, J. C. (2023). Implementación de Interfaz en Common. Madrid : Universidad Politécnica de Madrid .

Cruz, F. (2012). Importancia de la evaluación y autoevaluación en el rendimiento académico. México: Revista del Instituto.

García, F. (2023). ¿Por qué enseñar a aprender en la universidad? Madrid: Dialnet.

Jr., G. S. (1990). Common LISP. The Language. Second Edition. Massachusetts: Digital Press; 2nd Updated edición.

Keene, S. E. (1989). Object-Oriented Programming in COMMON LISP: A Programmer's Guide to CLOS. Addison-Wesley Professional.

Lamkins, D. (2004). Successful Lisp: How to Understand and Use Common Lisp. BookFix.com .

McCarthy, J. (1960). Recursive Functions of Symbolic Expressions. Massachusetts: Massachusetts Institute of Technology.

McCarthy, J. (1985). Programmer´s manual. Cambridge: Massrchusetts Institute of Toohnology.

Moyne, J. (2023). LISP. PRIMER LENGUAJE PARA PROGRAMADORES DE COMPUTADORAS.

Santos, M. (2019). La magia del triángulo: convivencia, conflicto e inclusión. Málaga: Contextos Educativos .

Seibel, P. (2005). Practical Common Lisp. Berkeley: Apress.

Shapiro, S. (1991). Common LISP: an interactive approach. New York: Library of Congress Cataloging-in-Publication Data.

Steele, G. (11990). Common LISP. The Language. Second Edition . Boston: Digital Press; 2nd Updated edición.

Touretzky, D. S. (1989). COMMON LISP:. California: Carnegie Mellon University.

Walrond, A. (2009). Practical Common Lisp - Distilled.

Watson, M. (2024). Loving Common Lisp, or the Savvy Programmer's Secret Weapon. McGraw-Hill.