UNIVERSIDAD AUTONOMA DE TAMAULIPAS

Ingeniería en Sistemas Computacionales

9° G

Programación de Sistemas de Base 2

(3:00PM – 4:00PM)

Informe final

Gutiérrez Garcés Edwin

Garcés Hernández Jesús Eduardo

Moreno Reyes Luis Daniel

Ruiz Yáñez Aaron Alfonso

Guerrero Martínez Jesús

Cd. Madero, Tamaulipas a 2025

Contenido

1. Introducción

En el desarrollo de compiladores modernos, la integración de herramientas de inteligencia artificial, especialmente del procesamiento de lenguaje natural (NLP), se ha convertido en una tendencia innovadora que potencia significativamente la experiencia del usuario. Tradicionalmente, los compiladores validan el código fuente desde una perspectiva estrictamente sintáctica y semántica, pero rara vez explican el significado funcional de las instrucciones o brindan sugerencias claras ante errores. Al integrar modelos de NLP, se puede interpretar el contexto del código, generar anotaciones comprensibles y sugerir correcciones más humanas e intuitivas.

Este proyecto propone un traductor diseñado para un lenguaje personalizado de control de robots, capaz de interpretar instrucciones como MOVE_FORWARD, WAIT, SHUTDOWN, y asociarlas no solo con validez técnica, sino con una descripción funcional de lo que hace cada acción. Asimismo, valida tipos, estructuras de control (IF, WHILE) y uso correcto de variables. Pero más allá del análisis tradicional, el sistema detecta errores de contexto como tipos incompatibles o funciones mal escritas, y propone soluciones mediante un módulo de NLP. De esta forma, se transforma un compilador estático en una herramienta asistida que "entiende" mejor al usuario.

Para ello, se integran tecnologías clave: un análisis léxico y sintáctico construido en Python, un motor semántico con tabla de símbolos, un generador de código intermedio (basado en pseudocódigo de tres direcciones) y un componente de NLP basado en la API de inferencia de Hugging Face. Esta combinación permite detectar errores, anotar funcionalidades del robot y procesar comentarios escritos en lenguaje natural. Si bien por limitaciones del entorno gratuito los modelos preentrenados como CodeBERT o GPT-2 no pudieron responder activamente en tiempo real, el sistema está diseñado para ser compatible y fácilmente ampliable a entornos profesionales donde esa integración sea viable.

2. Desarrollo del Proyecto

2.1 Descripción del Lenguaje

El lenguaje personalizado implementado en este proyecto está orientado al control de un robot mediante una serie de instrucciones estructuradas que simulan un dominio específico (DSL). Entre las funcionalidades soportadas se incluyen movimientos (MOVE_FORWARD, TURN_LEFT), manipulación de sensores (READ_SENSOR, ACTIVATE_SENSOR), operaciones de estado (SHUTDOWN, WAIT), y control de flujo mediante estructuras condicionales (IF) y cíclicas (WHILE).

El lenguaje reconoce tres tipos de datos: int, float y boolean, y permite declaraciones, asignaciones, comparaciones, expresiones aritméticas y booleanas, además de llamadas a funciones con o sin argumentos. Las instrucciones válidas pueden tener efectos funcionales como “el robot se moverá hacia adelante”, los cuales son anotados semánticamente.

2.2 Arquitectura del Sistema

El sistema se estructura como un pipeline clásico de compilación:

- Lexer (lexer.py): genera tokens a partir del archivo de entrada.
- Parser (parser.py): construye un AST validando estructura sintáctica.
- Semántico (semantico.py): verifica tipos, variables, funciones y anota significados funcionales.
- Código Intermedio (codigo_intermedio.py): convierte el AST en una representación de tres direcciones.
- Sugerencias NLP (sugerencias_nlp.py): procesa comentarios y errores con descripciones generadas.

2.3 Componentes Implementados

- lexer.py: utiliza expresiones regulares para identificar todos los tokens del lenguaje, incluyendo funciones, operadores, identificadores y delimitadores.
- parser.py: implementa un parser LL que reconoce declaraciones, asignaciones, estructuras IF/WHILE y llamadas a funciones válidas (con y sin argumentos).
- semantico.py: contiene la tabla de símbolos, valida tipos de datos, detecta errores como variables no declaradas o tipos incompatibles, y registra descripciones funcionales de las instrucciones.
- codigo_intermedio.py: genera instrucciones tipo tres direcciones como PARAM, CALL, STORE, facilitando la posible optimización.
- sugerencias_nlp.py: extrae los comentarios del código fuente y los convierte en anotaciones simuladas o reales con NLP.

2.4 Análisis Semántico

El analizador semántico es capaz de detectar errores complejos como:

- Uso de variables no declaradas.
- Asignaciones entre tipos incompatibles (ej. float a boolean).
- Condiciones no booleanas en IF/WHILE.
- Funciones mal escritas o mal utilizadas (ej. MOVE_FORWARD sin argumento).

Además, las funciones válidas se registran con descripciones legibles para humanos, integrando el significado funcional de cada instrucción en el análisis.

2.5 Generación de Código Intermedio

El generador traduce las acciones semánticas en una lista de instrucciones de código intermedio. Por ejemplo, una asignación se convierte en una operación tipo t1 = 5 seguida de STORE t1, x. Las funciones también generan parámetros (PARAM), llamadas (CALL) y transferencias (STORE). El resultado se guarda en un archivo codigo_intermedio.txt.

2.6 Procesamiento NLP (Hugging Face)

El sistema está diseñado para utilizar la Inference API de Hugging Face con modelos como CodeBERT o GPT-2. Aunque por limitaciones del entorno (Spaces gratuitos) la inferencia directa no fue funcional, el módulo está preparado para generar sugerencias explicativas con prompts como:

> ERROR: Variable 'z' usada sin declarar.
>
> SUGERENCIA: ¿Olvidaste declarar 'z'? Intenta: int z = 0;

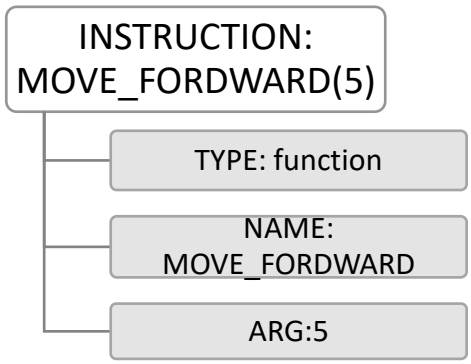
3. Diagramas y Flujos

3.1 Árbol Sintáctico con Anotaciones Semánticas

El árbol sintáctico generado por el analizador (AST) es una estructura interna representada como una lista de diccionarios que reflejan las instrucciones del lenguaje del robot. Cada nodo describe el tipo de instrucción (declaration, assign, if, while, function, etc.), los elementos involucrados y, cuando corresponde, sus argumentos y expresiones asociadas.

En este proyecto, cada instrucción reconocida por el parser se traduce a una estructura tipo diccionario que alimenta el análisis semántico. A partir de este árbol, el sistema no solo verifica tipos, sino que también anota el significado funcional de cada función reconocida.

Ejemplo:



Este AST es utilizado por el módulo semántico para validar tipos, verificar declaraciones, identificar errores y, además, generar anotaciones funcionales que describen la acción que realizará el robot.

Ejemplo simplificado de un AST en forma estructurada:

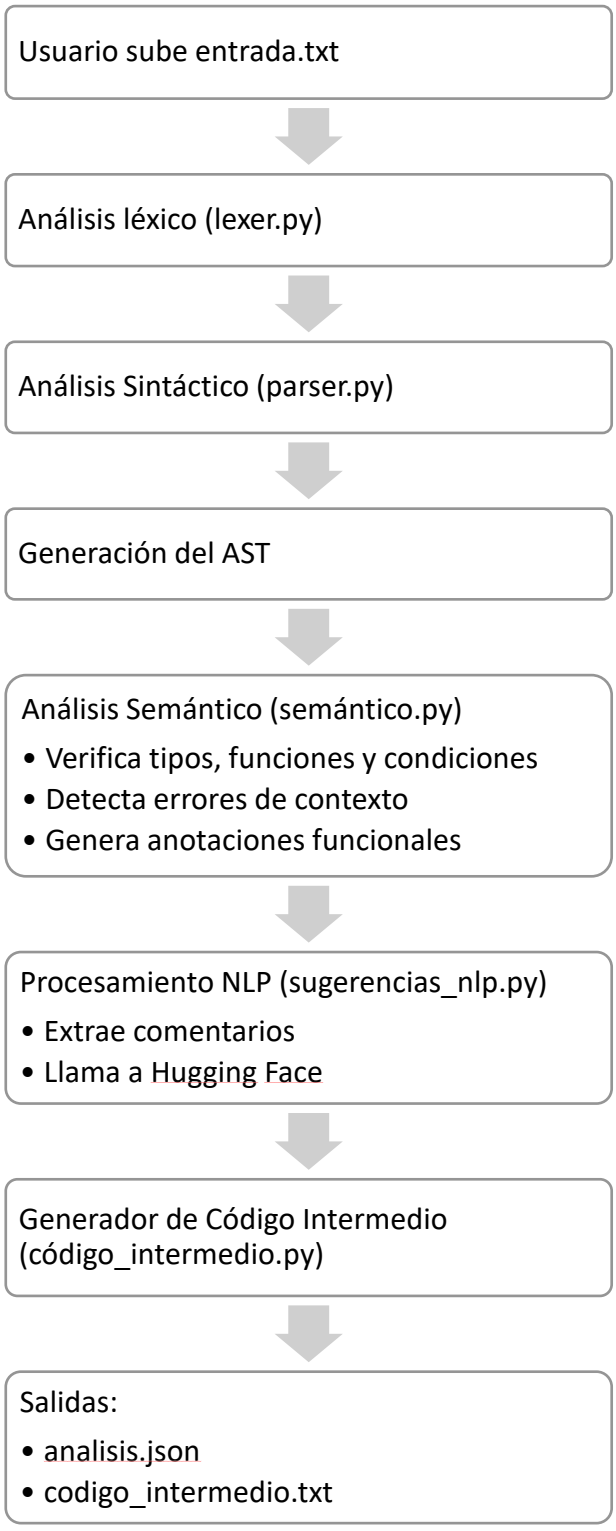
```
[
  {
    "type": "assign",
    "var": "x",
    "value": {"type": "string", "value": "hola"}
  },
  {
    "type": "function",
    "name": "MOVE_FORWARD",
    "arg": {"type": "num", "value": "5"}
  }
]
```

Cada una de estas instrucciones será evaluada por el analizador semántico para determinar su validez, y en el caso de funciones válidas, se agregará una anotación como:

MOVE_FORWARD: El robot se moverá hacia adelante

3.2 Flujograma del Proceso de Compilación con Interacción de Hugging Face

El siguiente flujo describe el proceso completo desde la carga del archivo de entrada hasta la generación de salidas analizadas, con intervención del modelo NLP:



Este proceso garantiza que el sistema no solo detecte errores y valide sintaxis, sino que también interprete el código desde una perspectiva funcional, aportando sugerencias y descripciones generadas de forma automatizada o simulada, según disponibilidad del modelo NLP.

4. Pruebas

Para validar el funcionamiento del compilador con integración NLP, se realizaron pruebas con archivos .txt que contienen distintos fragmentos de código escritos en el lenguaje del robot. Se evaluaron tanto ejecuciones exitosas como casos que intencionalmente contienen errores para probar la robustez del sistema y la utilidad de las sugerencias generadas.

4.1 Casos de Éxito

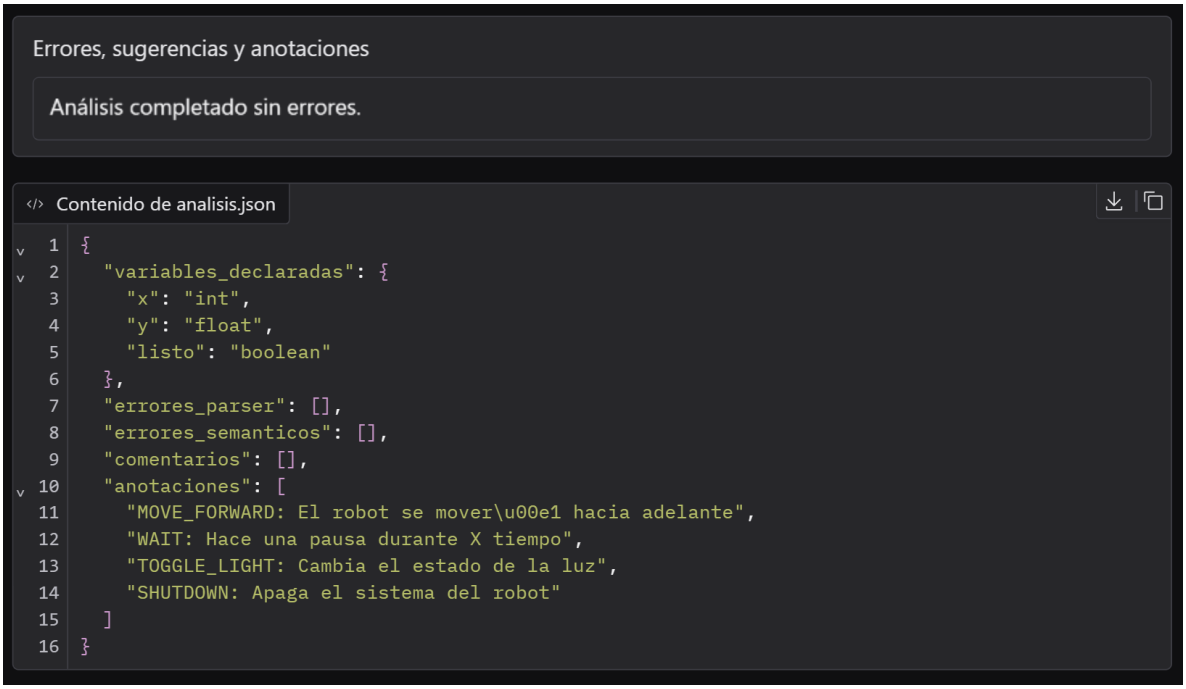
Entrada de prueba:

```
int x;
float y;
boolean listo;

x = 10;
y = 3.5;
listo = TRUE;

MOVE_FORWARD(5);
WAIT(2);
TOGGLE_LIGHT;
SHUTDOWN;
```

Resultado:



No hay errores léxicos, sintácticos ni semánticos. Las variables están bien declaradas y se muestran las anotaciones de lo que está haciendo el robot mediante las funciones, como por ejemplo:

MOVE_FORWARD: El robot se moverá hacia adelante

4.2 Manejo de Errores

Se evaluó el sistema con una entrada que contiene múltiples errores semánticos relacionados con:

- Tipos incompatibles
- Variables no declaradas
- Funciones mal formadas o con paréntesis incorrectos

Ejemplo de entrada:

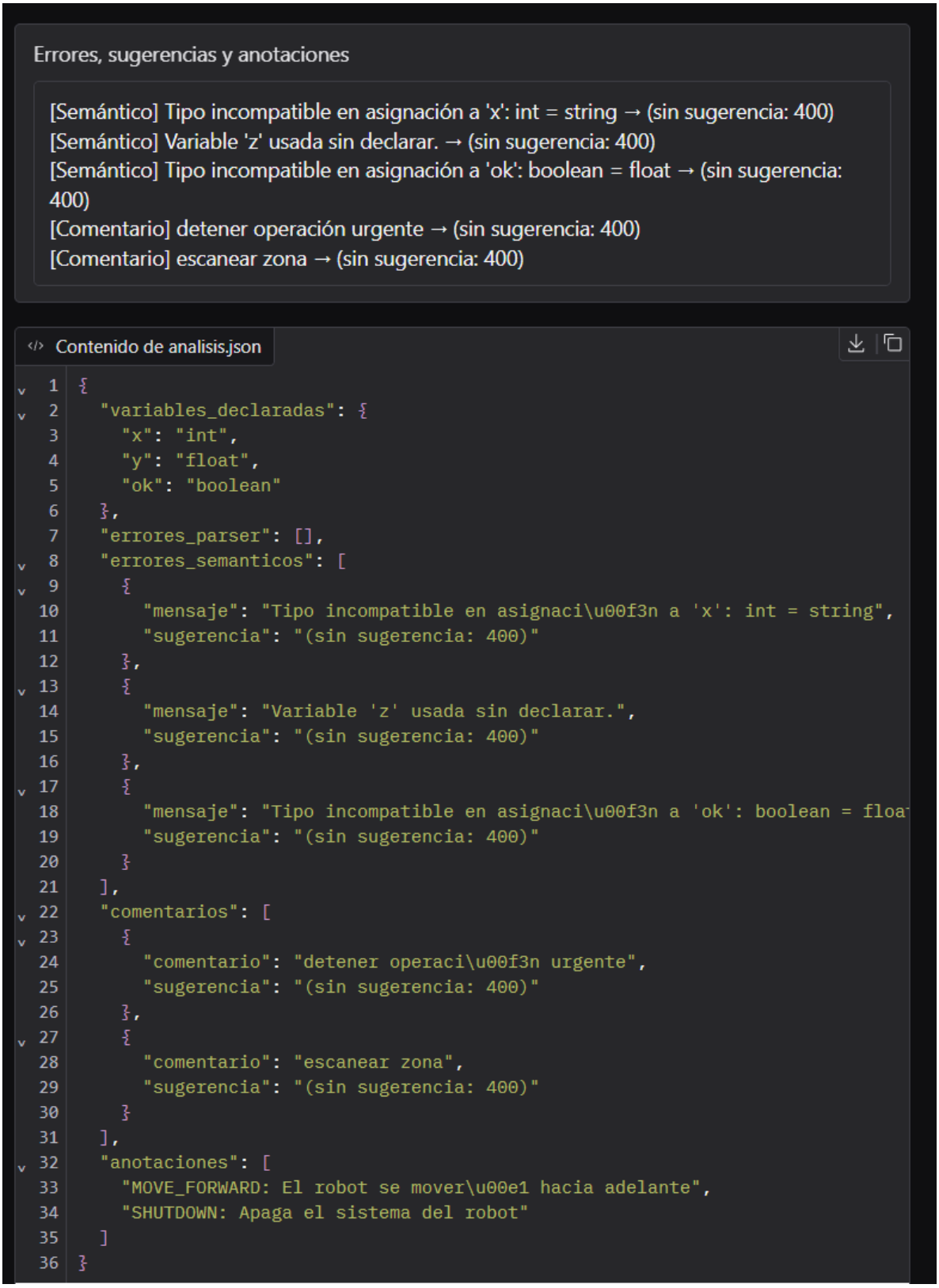
```
int x;
float y;
boolean ok;

x = "hola";
z = 10.5;
ok = 3.14;

MOVE_FORWARD(5);
SHUTDOWN;

// detener operación urgente
// escanear zona
```

Resultado:


Errores, sugerencias y anotaciones
[Semántico] Tipo incompatible en asignación a 'x': int = string → (sin sugerencia: 400)
[Semántico] Variable 'z' usada sin declarar. → (sin sugerencia: 400)
[Semántico] Tipo incompatible en asignación a 'ok': boolean = float → (sin sugerencia: 400)
[Comentario] detener operación urgente → (sin sugerencia: 400)
[Comentario] escanear zona → (sin sugerencia: 400)
Contenido de analisis.json
{
"variables_declaradas": {
"x": "int",
"y": "float",
"ok": "boolean"
},
"errores_parser": [],
"errores_semanticos": [
{
"mensaje": "Tipo incompatible en asignaci\u00f3n a 'x': int = string",
"sugerencia": "(sin sugerencia: 400)"
},
{
"mensaje": "Variable 'z' usada sin declarar.",
"sugerencia": "(sin sugerencia: 400)"
},
{
"mensaje": "Tipo incompatible en asignaci\u00f3n a 'ok': boolean = floa
"sugerencia": "(sin sugerencia: 400)"
}
],
"comentarios": [
{
"comentario": "detener operaci\u00f3n urgente",
"sugerencia": "(sin sugerencia: 400)"
},
{
"comentario": "escanear zona",
"sugerencia": "(sin sugerencia: 400)"
}
],
"anotaciones": [
"MOVE_FORWARD: El robot se mover\u00e1 hacia adelante",
"SHUTDOWN: Apaga el sistema del robot"
]
}

El sistema identifica correctamente todos los errores semánticos, con mensajes claros y detallados.

Se generan sugerencias explicativas, aunque el proyecto funciona y está listo para dar sugerencias mediante API a un modelo como GPT-2, Huggin Face en su plan gratuito no permite que las peticiones del proyecto al modelo, por tanto, en la sugerencia muestra: “(sin sugerencia: 400)”.

Los comentarios del código son extraídos y procesados, registrándose en el archivo de análisis como anotaciones NLP, incluso cuando el modelo externo no responde.

4.3 Evaluación del Impacto de los Modelos en la Experiencia del Usuario

El componente de NLP en este proyecto tiene un impacto directo en la claridad y utilidad del sistema para el usuario final. A diferencia de compiladores tradicionales que sólo reportan errores crudos, este sistema:

- Intenta explicar el porqué de los errores, usando lenguaje natural.
- Procesa comentarios como:

  ```
  // girar a la izquierda  - "Sugerencia: revisa la intención de 'girar a la izquierda'"
  ```

- Aunque el modelo de Hugging Face no pudo generar texto en tiempo real (por límites del entorno gratuito), la arquitectura del sistema ya está preparada para emplear respuestas reales al integrar un modelo como GPT-2 o CodeBERT en un entorno profesional.

Esto permite que el proyecto no sólo detecte errores, sino que actúe como un asistente de programación para el usuario, mejorando la comprensión del lenguaje y su propósito funcional.

6. Reflexión Final

El desarrollo de este proyecto representó una experiencia integral en la construcción de un compilador funcional, pero también una oportunidad para explorar nuevas fronteras al integrar herramientas de inteligencia artificial, como el procesamiento de lenguaje natural (NLP), dentro de un flujo de análisis semántico tradicional.

A nivel técnico, uno de los principales retos fue la correcta estructuración del lenguaje personalizado (DSL) para controlar un robot, garantizando que cada instrucción fuese interpretada no solo desde una perspectiva sintáctica, sino también con un propósito semántico claro. Este enfoque permitió validar acciones como MOVE_FORWARD(5) con contexto funcional y generar anotaciones que simulan una comprensión del propósito del código por parte del compilador.

Uno de los aspectos más valiosos fue el intento de integrar un modelo de Hugging Face, lo que trajo desafíos importantes. Si bien el entorno gratuito de Hugging Face Spaces impidió realizar inferencias NLP en tiempo real, el sistema fue diseñado modularmente para aceptar dichas integraciones en cuanto estén disponibles. Esta arquitectura permitió simular sugerencias basadas en reglas, lo cual resultó efectivo para fines demostrativos y validación de flujo.

Durante el proceso, se fortalecieron habilidades clave: desde el manejo de gramáticas, análisis semántico y generación de código intermedio, hasta la integración con APIs modernas y el diseño de una interfaz funcional en un entorno desplegado. Además, se reforzó la importancia de diseñar sistemas tolerantes a errores y preparados para escalar, aun cuando operen inicialmente bajo ciertas restricciones.

Como punto de mejora, en versiones futuras se propone incorporar almacenamiento de historial de análisis, permitir entrada directa desde interfaz de texto y, sobre todo, implementar la inferencia real con modelos como CodeBERT o T5, lo que dotaría al sistema de una capacidad aún más rica de interacción y aprendizaje contextual.

En conclusión, este proyecto logró materializar un compilador semántico enriquecido con NLP, ofreciendo una interfaz más comprensible, educativa y funcional para usuarios que desean programar instrucciones para robots, con validación y asistencia contextual automatizada.