



**UNIVERSIDAD ANDRÉS BELLO**

**FACULTAD DE INGENIERÍA**

**INGENIERÍA CIVIL INFORMÁTICA**

**TRANSFORMERS EN LA VECTORIZACIÓN DE CÓDIGO FUENTE PARA EL  
PROCESO DE APRENDIZAJE DE ESTUDIANTES EN CURSOS DE  
PROGRAMACIÓN**

Tesis para optar al título de Ingeniería Civil informática

**ESTUDIANTE:**

Joaquín Ignacio Mujica Villalobos

**Profesor guía:**

**Pablo Hernán Schwarzenberg Riveros**

**SANTIAGO – CHILE**

**ABRIL – 2025**

## **Agradecimientos**

*Quiero agradecer a todas las personas que me brindaron apoyo en este proceso y estuvieron conmigo entendiendo la presión y el peso del trabajo universitario.*

*A las amistades que pude obtener dentro de la universidad, las cuales fueron cruciales para crecer tanto como estudiante, a pesar de no haber estado presente gran parte del tiempo, las gratas memorias nunca se van a olvidar y sin duda son de las mejores experiencias que obtuve.*

*A mi profesor y director de carrera Pablo Schwarzenberg quien con sus consejos y gran paciencia pude seguir con mi trayecto universitario y este proyecto de título.*

*A mi círculo de amistades cercanas, quienes nos conocemos desde la época escolar y han sabido de primera mano cada complicación en la que necesite ser escuchado. Con una mención especial a mis amigas Francisca y Javiera quienes han visto desde mi peor hasta mi mejor versión a lo largo de mi vida universitaria.*

*Y por último a mi círculo familiar, con quienes a pesar de haberme visto en todos los estados de animo posibles, siempre confiaron y creyeron en mis capacidades, motivándome cada día a seguir adelante, a seguir aprendiendo y a seguir creciendo sin perder ninguno de los valores que me inculcaron.*

## Índice General

Índice de Figuras .....	2
Abstract .....	4
Resumen .....	5
1. Introducción.....	0
2. Revisión Bibliográfica.....	1
3. Objetivos del Trabajo .....	3
4. Metodología.....	3
5. Carta Gantt.....	5
6. Desarrollo .....	0
7. Resultados.....	7
8. Discusión. ....	30
9. Conclusiones.....	32
10. Limitaciones del trabajo y trabajos futuros .....	34
11. Referencias.....	35

## Índice de Figuras

Figura 5.1: “Carta Gantt del Proyecto” .....	5
Figura 6.1: “Carpeta Desarrollo del Proyecto”.....	2
Figura 6.2: “Código Fuente del Desarrollo – Parte 1” .....	3
Figura 6.3: “Fragmento 1 de Código Fuente del Desarrollo – Parte 2” .....	4
Figura 6.4: “Fragmento 2 de Código Fuente del Desarrollo – Parte 2” .....	5
Figura 6.5: “Fragmento 3 de Código Fuente del Desarrollo – Parte 2” .....	6
Figura 7.1: “Gráfico t-SNE Ejercicio 1 – Hito 1” .....	8
Figura 7.2: “Gráfico t-SNE Ejercicio 1 – Hito 1 (Ampliado)” .....	9
Figura 7.3: “Gráfico t-SNE Ejercicio 2 – Hito 1” .....	9
Figura 7.4: “Gráfico t-SNE Ejercicio 2 – Hito 1 (Ampliado)” .....	10
Figura 7.5: “Gráfico t-SNE Ejercicio 3 – Hito 1” .....	11
Figura 7.6: “Código Fuente Solución de Ejercicio 1 – Hito 1” .....	12

Figura 7.7: “Código Fuente Estudiante 1 de Ejercicio 1 – Hito 1” .....	13
Figura 7.8: “Código Fuente Estudiante 2 de Ejercicio 1 – Hito 1” .....	13
Figura 7.9: “Código Fuente Solución de Ejercicio 2 – Hito 1” .....	14
Figura 7.10: “Código Fuente Estudiante 1 de Ejercicio 2 – Hito 1” .....	15
Figura 7.11: “Código Fuente Estudiante 2 de Ejercicio 2 – Hito 1” .....	16
Figura 7.12: “Código Fuente Solución de Ejercicio 3 – Hito 1” .....	17
Figura 7.13: “Código Fuente Estudiante 1 de Ejercicio 3 – Hito 1” .....	18
Figura 7.14: “Código Fuente Estudiante 2 de Ejercicio 3 – Hito 1” .....	19
Figura 7.15: “Gráfico t-SNE de Ejercicio 1 – Hito 2” .....	20
Figura 7.16: “Gráfico t-SNE de Ejercicio 2 – Hito 2” .....	21
Figura 7.17: “Gráfico t-SNE de Ejercicio 3 – Hito 2” .....	22
Figura 7.18: “Código Fuente Solución de Ejercicio 1 – Hito 2” .....	23
Figura 7.19: “Código Fuente Estudiante 1 de Ejercicio 1 – Hito 2” .....	24
Figura 7.20: “Código Fuente Estudiante 2 de Ejercicio 1 – Hito 2” .....	25
Figura 7.21: “Código Fuente Solución de Ejercicio 2 – Hito 2” .....	25
Figura 7.22: “Código Fuente Estudiante 1 de Ejercicio 2 – Hito 2” .....	26
Figura 7.23: “Código Fuente Estudiante 2 de Ejercicio 2 – Hito 2” .....	26
Figura 7.24: “Código Fuente Solución de Ejercicio 3 – Hito 2” .....	27
Figura 7.25: “Código Fuente Solución Base de Ejercicio 3 – Hito 2” .....	28
Figura 7.26: “Código Fuente Estudiante 1 de Ejercicio 3 – Hito 2” .....	29
Figura 7.27: “Código Fuente Estudiante 2 de Ejercicio 3 – Hito 2” .....	30

## Abstract

It's common for students enrolled in degree programs that include programming in their study plans to encounter complicated challenges. Many students enter engineering or similar fields in higher education without prior experience or either practice in programming languages. As a result, instructors often face a wide spectrum of students with varying skills and levels of proficiency in the learning process. This scenario becomes even more complex when some students exhibit lower motivation due to multiple factors, making it more difficult for instructors to analyze individual needs and provide targeted support too.

Therefore, this Project aims to research and explore ways to implement a Code Vectorization and Machine Learning system that offers one or several support tools for instructors, enabling them for a better visualization and monitor each student's performance throughout applied programming courses. The system is designed to be based on Transformers Architecture, given its revolutionary role in the natural language processing, making it fundamental to this Project. Through this approach, the Project seeks to provide effective solutions, addressing the challenges posed by the diversity of students capabilities in programming education.

## Resumen

Es común que estudiantes de carreras que incluyan programación en su plan de estudio se encuentren con un desafío complejo de superar, muchos ingresan a campos de ingeniería o similares en la educación superior sin experiencia ni práctica previa en lenguajes de programación. Esto deriva en que los docentes se encuentren con un amplio espectro de estudiantes con diversas capacidades y niveles de habilidad para el proceso de aprendizaje, más aún cuando algunos de estos se encuentran con menos motivación por diversos motivos, y es por esto, que se presenta un desafío para los docentes ya que se les hace más difícil analizar y entregar el apoyo necesario a cada estudiante. Por lo que este proyecto tiene como objetivo investigar y encontrar maneras de implementar un sistema de Vectorización de Código, Aprendizaje Automático, que entregue una o varias vías de apoyo a los docentes para que visualicen el desempeño de cada estudiante durante el desarrollo de los cursos de programación aplicada, con el desafío de que el sistema este basado en la arquitectura Transformers, ya que esta corresponde a un tipo de arquitectura revolucionaria en cuanto al procesamiento de lenguaje natural, haciéndola fundamental para este proyecto. Con esto se espera que el proyecto proporcione soluciones efectivas para enfrentar el desafío del estudio de la diversidad de capacidades de los estudiantes de cursos de programación.

## 1. Introducción

La programación se ha convertido en un método fundamental para desarrollarse en este mundo digitalizado y son más los estudiantes que se interesan en este campo de la educación superior, pero en esta área académica poseer conocimientos previos de esta materia antes de ingresar a una carrera basada en la programación es una ventaja importante que genera una brecha donde se diferencian fácilmente quienes tienen experiencia, quienes pueden comprender fácilmente los lenguajes de programación, y quienes no tienen alguna de estas dos habilidades y son quienes comienzan a dar sus primeros pasos, por lo que la diversidad del estado de los estudiantes en los cursos es notable.

Los docentes están en constante necesidad y búsqueda de herramientas que les facilite la realización de clases y la aplicación de material de práctica a los estudiantes, más aún cuando esto se encuentran en un alto abanico de variedad de sus habilidades, y este campo no está exento de esta situación, al contrario, los cursos de programación corresponden a una constante aplicación práctica para que el estudiante se familiarice con las metodologías de trabajo, pero esto llega a ser perjudicial para los casos donde el estudiante no consigue seguir el ritmo de la clase, quedando atrás y/o frustrándose durante el aprendizaje. Para esto es que el docente debe estar disponible, el apoyo es fundamental para que el estudiante pueda comprender los problemas y dificultades que tenga, superándolos y aprendiendo en el proceso.

Es por esto mismo que quienes imparten estas materias deben ser capaces de identificar y entregar el apoyo a quienes lo necesitan más que otros, pero realizar esta tarea en tiempo real tiene una alta dificultad considerando la cantidad de estudiantes que puede llegar a tener a cargo en un curso de programación, esto se extiende a un más si es que las dificultades no se afrontan y el estudiante se encuentra con una barrera de aprendizaje que se ha acumulado en un tiempo prolongado, y si es posible sumar más a esto, las complicaciones que puede tener un estudiante son más de las esperadas, ya que los malentendidos pueden ser desde abstractos hasta un error mecánico por falta

de práctica, y en la mayoría de los casos es prácticamente imposible estudiar a mano las respuestas y códigos de cada estudiante en escalas de cursos de educación superior. Debe existir una manera de poder solucionar esto, lograr una manera de monitorear a cada estudiante, y es ese el objetivo de este proyecto, encontrar una vía de desarrollo que sea capaz de entregar el apoyo a los docentes permitiéndoles visualizar el desempeño de los estudiantes para identificar a quienes estén teniendo dificultades en la realización de actividades y cual es este mismo. Esto mediante un sistema de Vectorización de Código basado en la arquitectura Transformers, donde el proyecto se basará en su capacidad de procesamiento del lenguaje natural, adaptándose de manera efectiva a cada estructura y semántica del código de cada estudiante. El sistema en general utilizará las técnicas de aprendizaje automático para proporcionar el apoyo a los docentes, al entregarles información más detallada sobre el desempeño de los estudiantes, buscando mejorar la enseñanza en los cursos de programación en la educación superior y contribuyendo al éxito académico de cada estudiante.

En las siguientes secciones de este artículo, se estará detallando la metodología utilizada, comenzando por la inclusión de los artículos de referencia para este proyecto, el desarrollo empleado, los resultados esperados y las implicaciones de esta investigación de la enseñanza de cursos de programación.

## **2. Revisión Bibliográfica**

En esta sección se revisarán las investigaciones tanto previas como en curso, las cuales se relacionan de una manera directa o indirecta con la identificación de problemas comunes en respuestas de códigos de estudiantes en cursos de programación. Hasta este punto hay 3 fuentes clave que aportan conocimientos valiosos sobre este tema.

Dentro del desarrollo de las inteligencias artificiales, dos conceptos prominentes están tomando importancia como nuevos métodos para representar y comprender los datos: embeddings y transformers. Estos elementos pueden desempeñar un papel fundamental para los enfoques de la vectorización de código, proporcionando una manera clara de capturar la complejidad estructural de múltiples trabajos y tareas de programación estudiantiles.



En el ámbito del aprendizaje automático, los embeddings son representaciones numéricas para objetos complejos, ya sean tanto palabras como fragmentos de código, estando así diseñados para obtener las relaciones semánticas de forma eficiente. Entonces al aplicar este concepto a la vectorización de código, los embeddings desempeñan un papel crucial al permitir que algoritmos y modelos puedan manipular tanto estructura como significado de un código desarrollado por algún estudiante.

La importancia de integrar el uso de embeddings y Transformers en la vectorización de código radica en la capacidad para ofrecer representaciones contextualmente correctas. Esta combinación no solo potencia la precisión de tareas, sino que además facilita la comprensión de las complejidades de los códigos, así aportando a los análisis dentro de entornos educativos.

Un ejemplo claro de la aplicación de los embeddings en el ámbito de la vectorización de código es el estudio de Zhang, Chen y Oney en “VizProg: Identifying Misunderstanding by Visualizing Students’ Coding Process”. En este trabajo, se emplean embeddings para mapear el proceso de codificación de los estudiantes en un espacio euclidiano bidimensional. Cada acción y decisión en el código se representa mediante embeddings, permitiendo una visualización en tiempo real del progreso y enfoque de los estudiantes. (Zhang, A., 2023. VizProg).

Otro ejemplo de la combinación de embeddings y transformers para el análisis de código fuente es la obra de Azcona, Hsiao, Arora y Smeaton en “User2code2vec: Embedding for Profiling Students based on Distributional Representation of Source Code”. En este trabajo, se enfatiza la utilización de embeddings junto con transformers para perfilar a los estudiantes según sus estilos de programación. La habilidad de los transformers para capturar relaciones a largo plazo y dependencias en el código agrega profundidad al análisis. (Azcona, D., 2019. User2code2vec).

Además de los ejemplos anteriores, Rakhmanov en “A Comparative Study on Vectorization and Classification Techniques in Sentiment Analysis to Classify Student-Lecturer Comments” implementa embeddings en el análisis de sentimientos, demostrando su aplicabilidad tanto al contenido del código como a los comentarios interactivos, subrayando su versatilidad en diversos contextos académicos. (Ochilbek R., 2020. A Comparative Study on Vectorization and Classification Techniques in Sentiment Analysis to Classify Student-Lecturer Comments).

De esta manera, los embeddings y transformers se establecen como elementos esenciales para comprender la complejidad de cada código de programación, entregándonos valiosas herramientas para entender y analizar el desempeño de los estudiantes. Estos conceptos, ejemplificados por investigaciones como VizProg, User2code2vec y el Estudio de Técnicas en Análisis de Sentimientos, marcan pautas innovadoras en la enseñanza y comprensión de la programación en entornos educativos.

### **3. Objetivos del Trabajo**

Considerar un Objetivo General y dos objetivos específicos que contribuyan al logro del objetivo general. Hay que considerar que cumplan el criterio SMART (Específico, Medible, Alcanzable, Relevante, acotado en tiempo).

- Objetivo General: Evaluar factibilidad de usar vectorizaciones para Construir un Modelo de Habilidad de los estudiantes en base al código fuente que entregan en una plataforma online.
- Objetivo Específico 1: Construir un dataset con notas y código fuente para un conjunto de estudiantes de programación.
- Objetivo Específico 2: comparar las vectorizaciones de código generadas por modelos pre entrenados sobre el dataset.

### **4. Metodología.**

Se propone una metodología de trabajo basada en la recopilación de un conjunto de datos representativo en variabilidad de trabajos realizados por estudiantes, principalmente en el lenguaje Python, pero con opciones para recopilar también un conjunto que represente trabajos realizados en el lenguaje C++. Esto con la finalidad de convertirlas en una fuente a evaluar y comparar los

diversos modelos de Vectorización de Código propuestos en cada fuente clave de la revisión bibliográfica.

- **Recolección de Datos:** Normalizar y limpiar los códigos para eliminar posibles sesgos y ruido.
  - Incluir tareas con variedad dificultad y complejidad, para reflejar la variedad de la capacidad de los estudiantes.
- **Preprocesamiento de Datos:** Separar datos en conjuntos de entrenamiento, validación y prueba.
  - Buscar cómo establecer formato para facilitar la comparación entre tareas y soluciones.
- **División de Conjunto de Datos:** División de Conjunto de Datos: Separar datos en conjuntos de entrenamiento, validación y prueba.
  - Garantizar que cada conjunto represente adecuadamente los trabajos de programación.
- **Implementación de Modelos:** Desarrollar modelos de vectorización de código que se basen en las arquitecturas mencionadas en las fuentes clave (VizProg, User2code2vec, Sentiment Analysis).
  - Investigar si es necesario adaptar los parámetros para optimizar el rendimiento de los modelos en función de los conjuntos de datos.
- **Evaluación Comparativa:** Evaluación Comparativa: Evaluar cada modelo usando métricas relevantes, como la precisión, capacidad de identificar errores y eficacia en tareas de análisis.
  - Comparar resultados obtenidos con los modelos de las fuentes clave.
- **Análisis e Interpretación:** Analizar los resultados para identificar fortalezas y debilidades de cada modelo.

- Interpretar los hallazgos en relación con las necesidades específicas de la enseñanza de la programación en la educación superior.

Esta metodología permitirá una evaluación ideal de los modelos de vectorización de código, proporcionando información valiosa para la selección y mejora de enfoques en la enseñanza de la programación.

## 5. Carta Gantt

Figura 5.1: “Carta Gantt del Proyecto”

Carta Gantt (Transformers en Vectorizacion deCodigo)																			
Hitos	N°	Tareas	ene-25					feb-25				mar-25				Abril			
			Semana 1	Semana 2	Semana 3	Semana4	Semana5	Semana 1	Semana 2	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4
1	1.1	Investigacion de Modelos Pre entrenados																	
	1.2	Recopilacion de Codigos Fuente																	
	1.3	Ordenamiento y Limpieza de Dataset																	
2	1.4	Revision de Integridad de Archivos																	
	1.5	Desarrollo Script - Parte 1																	
	1.6	Selección de Estudiantes para Hitos																	
3	2.1	Desarrollo Script - Parte 2																	
	2.2	Tokenizacion de Codigo Fuente con CodeBERT																	
	2.3	Filtrado de Tokens Esenciales																	
4	2.4	Generacion de Vectores																	
	2.5	Visualizacion de Graficos con t-SNE																	
	2.6	Ajustes Iterativos																	
5	3.1	Identificacion y Analisis de Estudiantes																	
	3.2	Evaluacion Visual de Resultados																	
	3.3	Redaccion de Informe																	
	3.4	Elaboracion de Graficos Finales																	

La figura 5.1 corresponde a la carta Gantt del proyecto mostrando la asignación de las diversas tareas provenientes de los puntos principales las cuales se desarrollaron a lo largo del Primer Semestre del año 2025, y estas mismas se agruparon en los siguientes Hitos de Desarrollo: Organización de la Base de Datos, Transformación de Archivos para Preprocesamiento, Tokenización y Extracción de Componentes Clave, y por último Vectorización y Análisis Dimensional. Para luego, al finalizar todos los procesos, realizar los análisis y conclusiones del proyecto.

## 6. Desarrollo

Con el objetivo de evaluar la capacidad de modelos pre entrenados basados en la arquitectura Transformers para representar de forma significativa el desarrollo de código por parte de estudiantes de cursos en la educación superior, se llevó a cabo un enfoque experimental dividido en etapas definidas.

- Organización de la Base de Datos

Se seleccionaron dos Hitos de trabajo (Denominados como Hito 1 e Hito 2) de entre las múltiples carpetas de Bandejas de Entrega de Proyectos de un curso de Programación de Educación Superior, cada uno de estos con tres carpetas de diferentes ejercicios prácticos de programación. De cada ejercicio se extrajo un conjunto de códigos entregados por los estudiantes, asegurando una cantidad representativa cercana a 50 entradas (50 archivos entregados por estudiantes). Se priorizó la repetición de estudiantes dentro de un mismo Hito para permitir análisis longitudinales dentro de estos mismos, mientras los identificadores de los estudiantes (IDs) son diferentes para los estudiantes entre Hitos, lo que permitió evaluar la capacidad del sistema para manejar disociación de identidades entre contextos.

- Transformación de Archivos para Preprocesamiento

Dado que el modelo entrenado CodeBERT requiere entradas de archivos en formato texto plano, se implementó un script automatizado que convierte los archivos originales entregados por los estudiantes en formato .py (Python) a archivos .txt, conservando así su contenido estructural y sintáctico para mantener la semántica de dichos códigos.

- Tokenización y Extracción de Componentes Clave

Cada archivo fue tokenizado utilizando la función RobertaTokenizer, adaptado para la comprensión del lenguaje de programación. A partir de esta tokenización, se identificaron los dos tokens que se presentan con mayor frecuencia en cada código (tanto en las entregas estudiantiles como en las soluciones de cada ejercicio), bajo la hipótesis de que representan componentes

semánticamente esenciales del desarrollo entregado. Se descartaron los tokens poco frecuentes restantes para mantener un foco claro en la estructura principal de cada entrega.

- Vectorización y Análisis Dimensional

Se generaron vectores de representación (embeddings) utilizando el modelo CodeBERT, extrayendo la media de los tensores ocultos como representación condensada del código completo. Los vectores luego fueron proyectados a un espacio bidimensional mediante un algoritmo t-SNE, lo cual permitió visualizar los agrupamientos o las dispersiones anteriormente planteadas entre códigos de estudiantes y soluciones. Esto facilitó la interpretación visual del rendimiento estructural y semántico por parte de del cuerpo docente y para efecto de análisis.

Este pipeline de trabajo fue desarrollado mediante dos scripts principales:

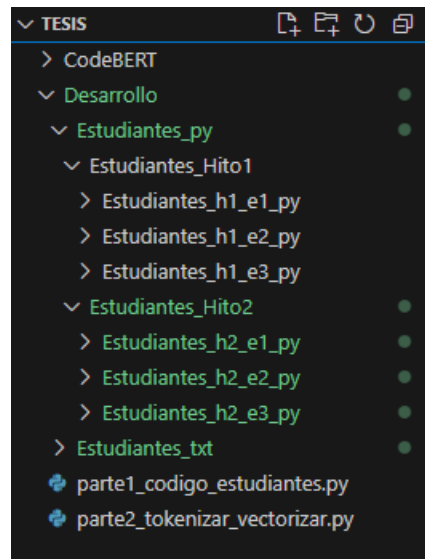
- a. Uno encargado de transformar las carpetas con archivos .py en los archivos .txt necesarios para sus procesamientos.
- b. Un segundo responsable de tokenizar, filtrar, vectorizar y graficar las representaciones estudiantiles en los espacios para su visualización comprensible.

Mediante etapas sucesivas alineadas directamente con los objetivos de metodológicos previamente identificados. A continuación, se presenta un resumen del flujo del trabajo ejecutado, desde la organización inicial de datos hasta la experimentación final:

- Organización y Limpieza del Dataset

Se estructuran las carpetas del dataset institucional original, separando los códigos fuente de los estudiantes por Hito y Ejercicio. Se priorizó la integridad de los archivos y la consistencia en los identificadores anteriormente mencionadas. Los datos fueron divididos en carpetas tipo Estudiante\_py/HitoX/EjercicioY, replicando la estructura lógica del curso.

Figura 6.1: “Carpeta Desarrollo del Proyecto”



La figura 6.1 muestra la organización de la carpeta para el ambiente de desarrollo del proyecto, donde se aprecia la presencia tanto de la carpeta con el modelo pre entrenado, la carpeta de Desarrollo que contiene las carpetas de los archivos de los estudiantes junto a las soluciones en el formato .py, y la carpeta de los archivos donde se envían al ser transformados a archivos .txt, además de ambos scripts correspondientes a las 2 partes del proyecto.

- Conversión Automatizada a Formato Procesable (Parte 1)

Se desarrolla un script (Parte 1) cuya función principal es iterar sobre las carpetas organizadas de archivos .py, tomar sus rutas de directorio y convertirlos en archivos .txt con el mismo contenido para enviarlas a su propia carpeta para su procesamiento. Este paso es esencial para preparar los archivos a su análisis con el modelo CodeBERT.

Figura 6.2: “Código Fuente del Desarrollo – Parte 1”



```
parte1_codigo_estudiantes.py X
Desarrollo > parte1_codigo_estudiantes.py > ...
1  from pathlib import Path
2
3  # Definir rutas de origen y destino
4  ruta_py = Path("C:/Users/joaqu/UNAB/Tesis/Desarrollo/Estudiantes_py/Estudiantes_Hito2/Estudiantes_h2_e3_py")
5  ruta_txt = Path("C:/Users/joaqu/UNAB/Tesis/Desarrollo/Estudiantes_txt/")
6  ruta_txt.mkdir(parents=True, exist_ok=True)
7
8  # Leer cada archivo y convertirlos de .py a .txt
9  for archivo in ruta_py.glob("*.py"):
10     contenido = archivo.read_text(encoding='utf-8')
11     nuevo_txt = ruta_txt / (archivo.stem + ".txt")
12     nuevo_txt.write_text(contenido, encoding='utf-8')
13     print(f"Convertido: {archivo.name} -> {nuevo_txt.name}")
14
```

La figura 6.2 corresponde a la parte 1 del script del desarrollo del proyecto, donde se aprecia el uso de ambos llamados a las rutas de directorio para capturar y transformar los archivos tanto de estudiantes como soluciones en formato .py para transformarlos a archivos .txt y enviarlos a su respectiva carpeta.

- Desarrollo del Script de Procesamiento (Parte 2)

Teniendo ya generada la carpeta de archivos .txt, se diseña un segundo script que tokeniza los contenidos, filtra los dos tokens más frecuentes por archivo, convierte dichos tokens en IDs, genera sus embeddings y finalmente grafica los resultados en un espacio bidimensional con el método t-SNE.

Cada entrega fue identificada con una etiqueta visual genérica, “Estudiante N”, y las soluciones con “Solución”, con el objetivo de identificar a los estudiantes de manera más rápida y mantener el foco en el análisis estructural del código.



Figura 6.3: “Fragmento 1 de Código Fuente del Desarrollo – Parte 2”

```
parte2_tokenizar_vectorizar.py X
Desarrollo > parte2_tokenizar_vectorizar.py > ...
1  from transformers import RobertaTokenizer, RobertaModel
2  from pathlib import Path
3  from collections import Counter
4  import torch
5  from sklearn.manifold import TSNE
6  from sklearn.decomposition import PCA
7  import matplotlib.pyplot as plt
8  import numpy as np
9
10 # Inicializar modelo y tokenizador de modelo pre entrenado
11 tokenizer = RobertaTokenizer.from_pretrained("microsoft/codebert-base")
12 model = RobertaModel.from_pretrained("microsoft/codebert-base")
13
14 # Preparar ruta de datos .txt
15 ruta_carpeta = Path("C:/Users/joacu/UNAB/Tesis/Desarrollo/Estudiantes_txt")
16 archivos_txt = list(ruta_carpeta.glob('*.txt'))
17
18 # Listas para almacenar Embeddings y Etiquetas de archivos de estudiantes
19 embeddings = []
20 etiquetas = []
21
22 # Lectura de carpeta con archivos .txt
23 if not archivos_txt:
24     print("No se encontraron archivos .txt en la ruta especificada.")
25 else:
26     for archivo_txt in archivos_txt:
27         try:
28             with open(archivo_txt, 'r', encoding='utf-8') as archivo:
29                 codigo = archivo.read()
30
31                 # Tokenizar codigo fuente
32                 tokens = tokenizer.tokenize(codigo)
33
34                 # Contar frecuencia de Tokens - Identificar los 2 mas esenciales
35                 frecuencia = Counter(tokens)
36                 tokens_esenciales = [tok for tok, _ in frecuencia.most_common(2)]
37
```

La figura 6.3 corresponde primer fragmento de la Parte 2 del script del desarrollo del proyecto. Donde se aprecia el uso de las librerías, la iniciación del modelo pre entrenado junto al tokenizador, el llamado a la ruta del directorio de los archivos .py ya transformados a archivos .txt y el inicio de la lectura de esta carpeta, en donde para cada archivo se le realiza su propia tokenización junto al conteo de su frecuencia de tokens, para identificar los 2 más esenciales.

Figura 6.4: “Fragmento 2 de Código Fuente del Desarrollo – Parte 2”

```
parte2_tokenizar_vectorizar.py X
Desarrollo > parte2_tokenizar_vectorizar.py > ...

30
31     # Tokenizar codigo fuente
32     tokens = tokenizer.tokenize(codigo)
33
34     # Contar frecuencia de Tokens - Identificar los 2 mas esenciales
35     frecuencia = Counter(tokens)
36     tokens_esenciales = [tok for tok, _ in frecuencia.most_common(2)]
37
38     # Condicional para codigos sin Tokens suficientes
39     if not tokens_esenciales:
40         print(f"No se encontraron tokens esenciales en {archivo_txt.name}")
41         continue
42
43     # Filtrar codigo Tokenizado y convertir los mas esenciales a IDs
44     tokens_filtrados = [tok for tok in tokens if tok in tokens_esenciales]
45     ids = tokenizer.convert_tokens_to_ids(tokens_filtrados)
46     if not ids:
47         print(f"No se pudieron generar IDs en {archivo_txt.name}")
48         continue
49
50     # Creacion de Tensores de entrada y Obtener Embeddings del modelo pre entrenado
51     entrada = torch.tensor([ids])
52     with torch.no_grad():
53         salida = model(entrada)
54         emb = salida.last_hidden_state.mean(dim=1).squeeze().numpy()
55         embeddings.append(emb)
56         etiquetas.append(archivo_txt.stem)
57
58 except Exception as e:
59     print(f"Error al procesar {archivo_txt.name}: {str(e)}")
60
```

La figura 6.4 corresponde al segundo fragmento de la Parte 2 del script del desarrollo del proyecto. En donde se muestra la condicional de alertar en caso de que algún archivo no contenga tokens suficientes para ser procesado, también una condicional en caso de que no se puedan generar IDs suficientes en base a los tokens existentes, y la creación de los tensores junto a la obtención de los Embeddings.

Figura 6.5: “Fragmento 3 de Código Fuente del Desarrollo – Parte 2”

```
parte2_tokenizar_vectorizar.py X
Desarrollo > parte2_tokenizar_vectorizar.py > ...

59     print(f"Error al procesar {archivo_txt.name}: {str(e)}")
60
61     # Solo si hay datos válidos, verificar generacion de Embeddings
62     if embeddings:
63         embeddings = np.array(embeddings)
64
65         # Metodo t-SNE
66         if len(embeddings) > 1:
67             tsne = TSNE(n_components=2, perplexity=min(5, len(embeddings) - 1), random_state=0)
68             emb_tsne = tsne.fit_transform(embeddings)
69
70             # Crear etiquetas tipo "Estudiante 1", "Estudiante 2", ..., "solucion"
71             etiquetas_visuales = []
72             contador = 1
73             for etq in etiquetas:
74                 if "solucion" in etq.lower():
75                     etiquetas_visuales.append("Solución")
76                 else:
77                     etiquetas_visuales.append(f"Estudiante {contador}")
78                     contador += 1
79
80             # Visualización de Grafico
81             plt.figure(figsize=(12, 8))
82             for i, nombre in enumerate(etiquetas_visuales):
83                 plt.scatter(emb_tsne[i, 0], emb_tsne[i, 1])
84                 plt.annotate(nombre, (emb_tsne[i, 0], emb_tsne[i, 1]), fontsize=10)
85
86             plt.title("t-SNE de Embeddings - Visualización por Estudiante")
87             plt.xlabel("Componente 1")
88             plt.ylabel("Componente 2")
89             plt.grid(True)
90             plt.show()
91
92             # Metodo PCA
93             if len(embeddings) > 1:
94                 pca = PCA(n_components=2)
95                 emb_pca = pca.fit_transform(embeddings)
96
97             else:
98                 print("No hay suficientes vectores para aplicar PCA.")
99         else:
100             print("No se generaron embeddings válidos.")
101
```

La figura 6.5 corresponde al último fragmento del script perteneciente a la Parte 2 del desarrollo del proyecto, en donde se muestra la aplicación del método t-SNE junto a la creación de las etiquetas para los puntos de los Estudiantes e identificarlos más rápidamente al momento de visualizar los gráficos, junto a condicionales para indicar insuficiencia de vectores para aplicar la generación de gráficos, o algún impedimento para generar Embeddings válidos.

- Iteraciones y Ajustes

Durante la implementación del segundo script, se realizan múltiples pruebas para validar que las dimensiones de los vectores fueran las correctas, que el número de estudiantes coincidiera con los archivos procesados y que el sistema generara resultados interpretables. Se resolvieron también múltiples errores relacionados a tensores vacíos, archivos sin contenido relevante y casos excepcionales donde los valores de archivos de estudiante no eran validados por los parámetros del modelo. Y también se ajustaron parámetros de t-SNE (como perplexity) para adaptarse al número de muestras.

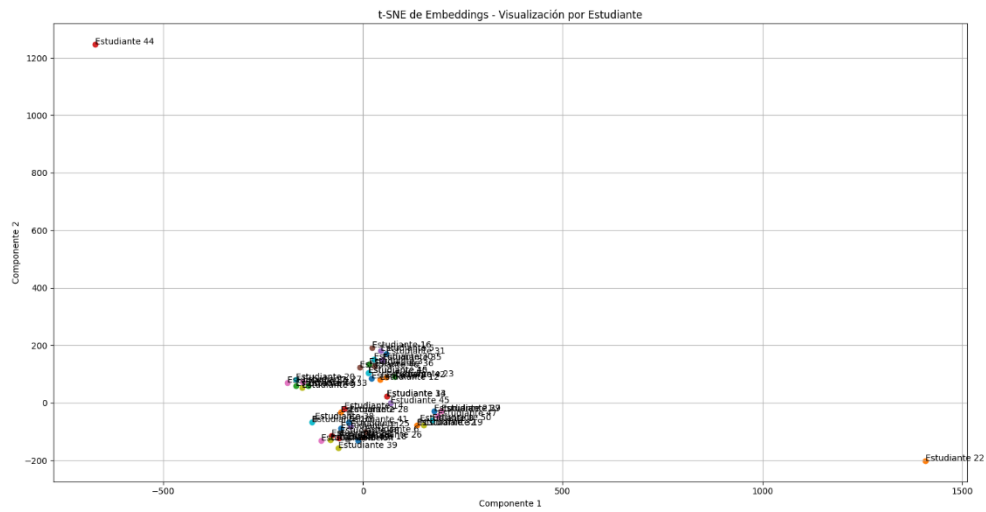
## **7. Resultados.**

En todos los gráficos realizados mediante el método t-SNE para cada Ejercicio dentro de ambos Hitos 1 y 2, cada punto representa una de las respuestas de cada estudiante mientras que el punto identificado como “Solución” corresponde al código correcto esperado por parte de los estudiantes. En todos los gráficos podemos observar agrupaciones que pueden interpretarse como similitudes estructurales o de lógica entre los códigos de los respectivos estudiantes.

- Análisis Hito 1

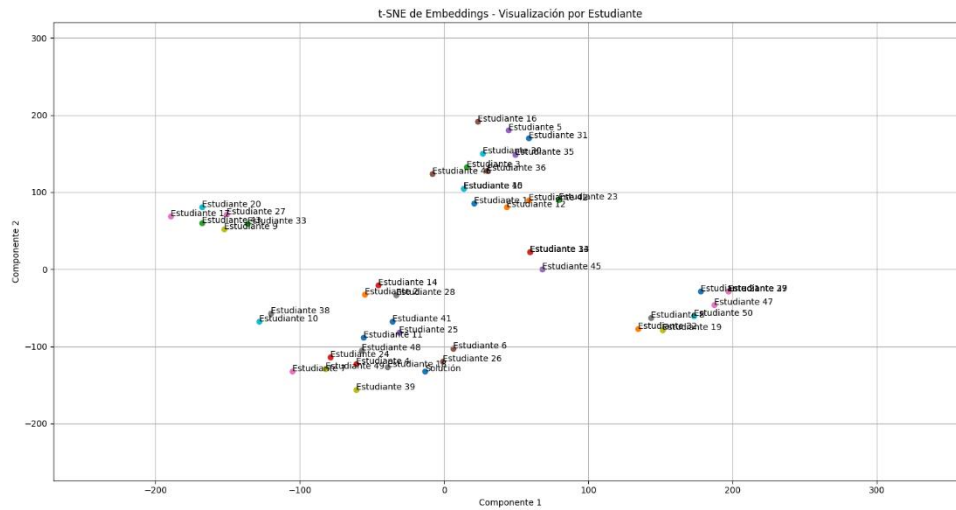
Para el Hito 1 del análisis, se generaron embeddings a partir de los códigos desarrollados por los estudiantes y las soluciones en los ejercicios correspondientes. En las figuras 1, 1 Ampliado, 2, 2 Ampliado y 3 se visualizan los resultados del agrupamiento de estos vectores en un espacio bidimensional utilizando el método t-SNE.

Figura 7.1: “Gráfico t-SNE Ejercicio 1 – Hito 1”



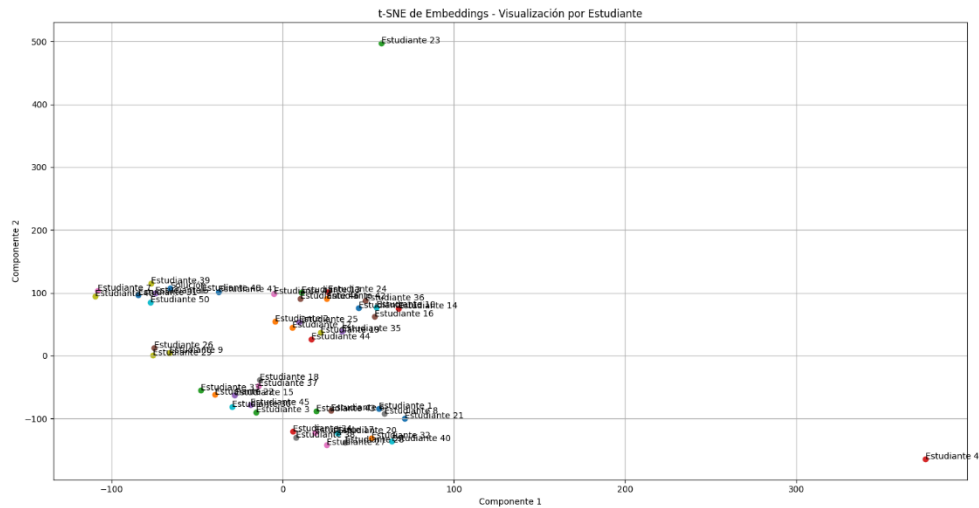
La figura 7.1 corresponde al gráfico t-SNE para el Ejercicio 1 del Hito 1, donde es visible la dispersión de datos en 2 estudiantes específicos mientras que el resto de los estudiantes se agrupan en varios sectores alrededor del punto Solución.

Figura 7.2: “Gráfico t-SNE Ejercicio 1 – Hito 1 (Ampliado)”



La figura 7.2 corresponde a una vista ampliada del grafico anterior para apreciar de mejor manera las agrupaciones principales de los estudiantes alrededor del punto Solución.

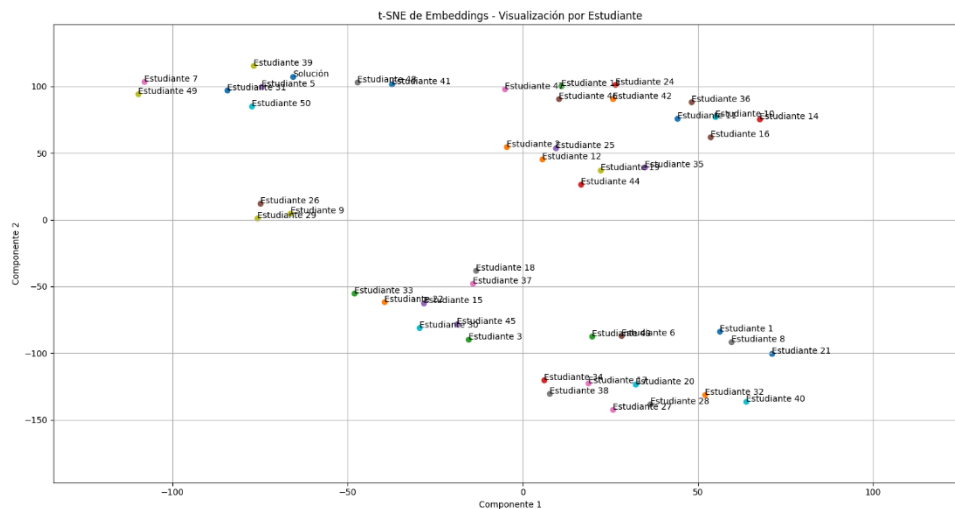
Figura 7.3: “Gráfico t-SNE Ejercicio 2 – Hito 1”



La figura 7.3 corresponde al grafico del Ejercicio 2 del Hito 1, en donde al igual que en el caso anterior, son visibles las agrupaciones principales de los estudiantes alrededor del punto Solución

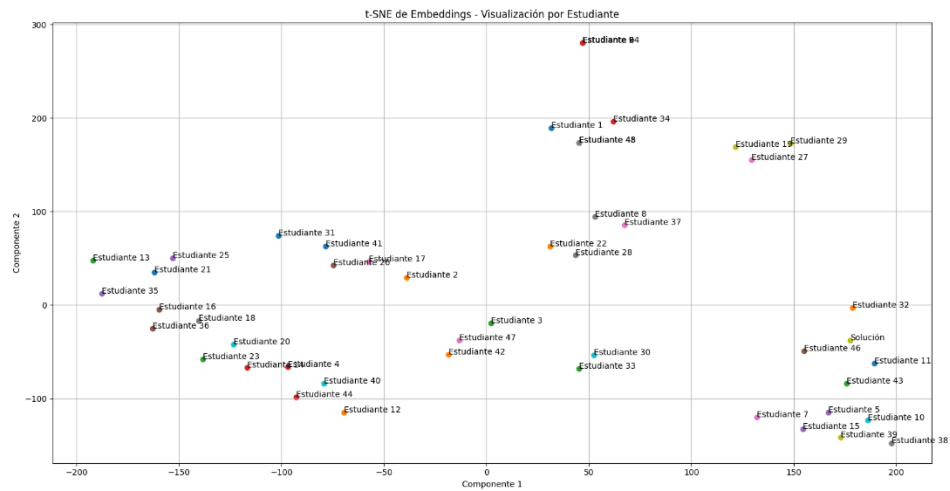
mientras que nuevamente se presenta la dispersión de datos con otros 2 estudiantes nuevos quienes con los componentes principales de sus respuestas se alejan del resto del curso.

Figura 7.4: “Gráfico t-SNE Ejercicio 2 – Hito 1 (Ampliado)”



La figura 7.4 es nuevamente una vista ampliada de la figura anterior, realizada para apreciar de mejor manera las principales agrupaciones de estudiantes en relación con el punto Solución.

Figura 7.5: “Gráfico t-SNE Ejercicio 3 – Hito 1”



La figura 7.5 corresponde al gráfico realizado para analizar los resultados del Ejercicio 3 del Hito 1, en donde a diferencia de los hitos anteriores, aquí no se presentan casos de dispersión de datos extremadamente alejados de las agrupaciones principales, a pesar de que existe un grupo significativamente grande en el sector inferior izquierdo, no hubo casos de estudiantes demasiado alejados en relación con el punto Solución.

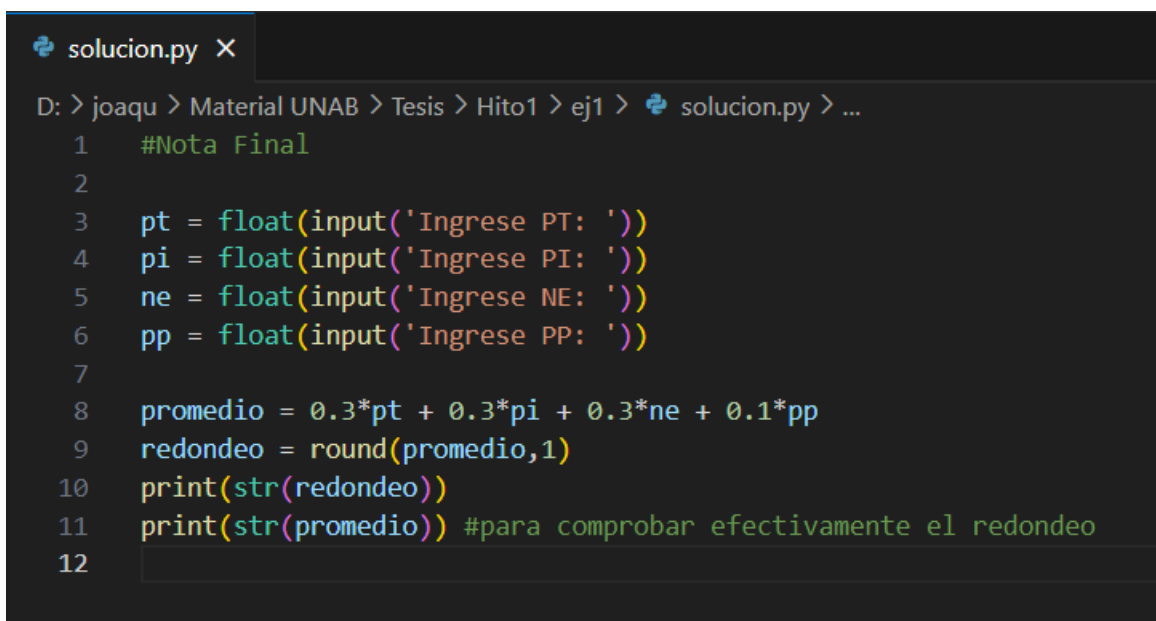
- Estudiantes Hito 1

Para estudiar casos de estudiantes específicos dentro del Hito 1, se eligen a 2 estudiantes, que deben estar presentes en todos los ejercicios, de entre una muestra de los primeros 50 dentro de cada Ejercicio en el Hito 1. Para el Estudiante 1 (archivo 0dad49841936a5216d722ade6bbad0fe), buscando su ID en los ejercicios dentro de Hito 1, para los ejercicios 1, 2 y 3, el programa lo identifica como “Estudiante 26” para Ejercicio 1, “Estudiante 22” para Ejercicio 2 y “Estudiante 22” nuevamente para Ejercicio 3, esto es debido a que su posición dentro de las 50 entradas de estudiantes varía por la naturaleza de que los estudiantes podían elegir si hacer o no los diversos ejercicios de cada Hito.



Para el Estudiante 2 (0f80d357fa095657efd74e1540915525), igualmente buscando su ID en los diferentes ejercicios dentro de Hito 1, para los ejercicios 1, 2 y 3 en los que está presente, el script lo identifica como “Estudiante 44” para el Ejercicio 1, “Estudiante 38” para el Ejercicios 2 y “Estudiante 36” para el Ejercicio 3, esto debido también a su posición dentro de las 50 entradas de estudiantes en la muestra elegida para analizar Hito 1.

Figura 7.6: “Código Fuente Solución de Ejercicio 1 – Hito 1”



```
solucion.py X
D: > joaqu > Material UNAB > Tesis > Hito1 > ej1 > solucion.py > ...
1  #Nota Final
2
3  pt = float(input('Ingrese PT: '))
4  pi = float(input('Ingrese PI: '))
5  ne = float(input('Ingrese NE: '))
6  pp = float(input('Ingrese PP: '))
7
8  promedio = 0.3*pt + 0.3*pi + 0.3*ne + 0.1*pp
9  redondeo = round(promedio,1)
10 print(str(redondeo))
11 print(str(promedio)) #para comprobar efectivamente el redondeo
12
```

La figura 7.6 corresponde al script de la Solución del Ejercicio 1 del Hito 1.

Figura 7.7: “Código Fuente Estudiante 1 de Ejercicio 1 – Hito 1”

```
hito1_ej1_0dad49841936a5216d722ade6bbad0fe.py X
D: > joaqu > Material UNAB > Tesis > Hito1 > ej1 > hito1_ej1_0dad49841936a5216d722ade6bbad0fe.py > ..
1  #Nota final
2  PT = float(input("Ingrese promedio de tareas: "))
3  PI = float(input("Ingrese promedio de interrogaciones: "))
4  NE = float(input("ingrese nota del examen: "))
5  PP = float(input("ingrese promedio de presentacion: "))
6
7  PF = (0.3*(PT))+(0.3*(PI))+(0.3*(NE))+(0.1*(PP))
8  a = round(PF,1)
9  print(a)
10 |
```

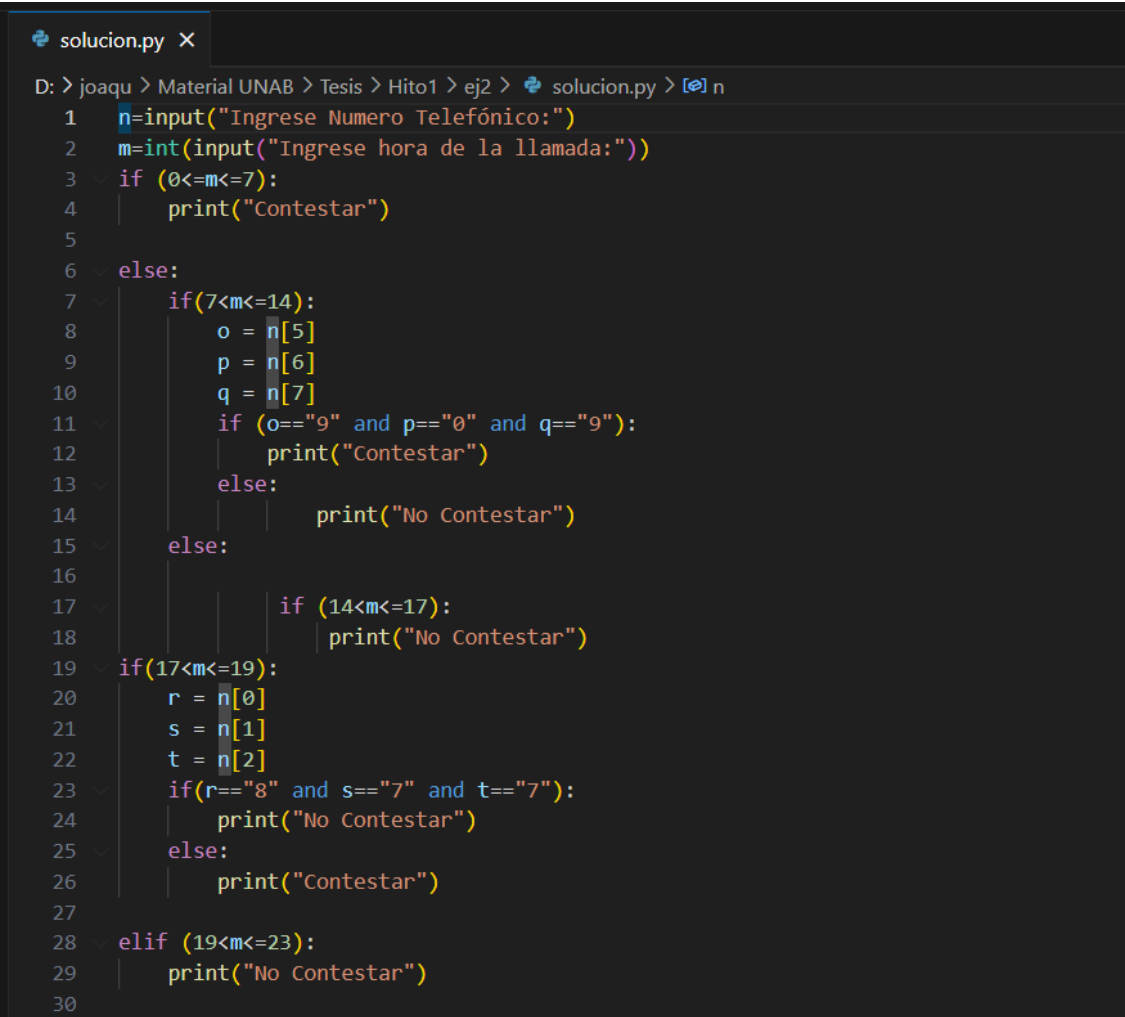
La figura 7.7 muestra el desarrollo del Estudiante 1 para el Ejercicio 1 del Hito 1.

Figura 7.8: “Código Fuente Estudiante 2 de Ejercicio 1 – Hito 1”

```
hito1_ej1_0f80d357fa095657efd74e1540915525.py X
D: > joaqu > Material UNAB > Tesis > Hito1 > ej1 > hito1_ej1_0f80d357fa095657efd74e1540915525.py > ...
1  #Nota final
2  PT = float(input('Ingresa nota de Tarea: '))
3  PI = float(input('Ingresa nota de Tarea: '))
4  NE = float(input('Ingresa nota de Tarea: '))
5  PP = float(input('Ingresa nota de Tarea: '))
6
7  Promedio = (0.3 * PT + 0.3 * PI + 0.3 * NE + 0.1 * PP)
8  Promedio_Redondeado = round(Promedio,1)
9  print(Promedio_Redondeado)
```

La figura 7.8 muestra el desarrollo entregado por el Estudiante 2 para el Ejercicio 1 del Hito 1.

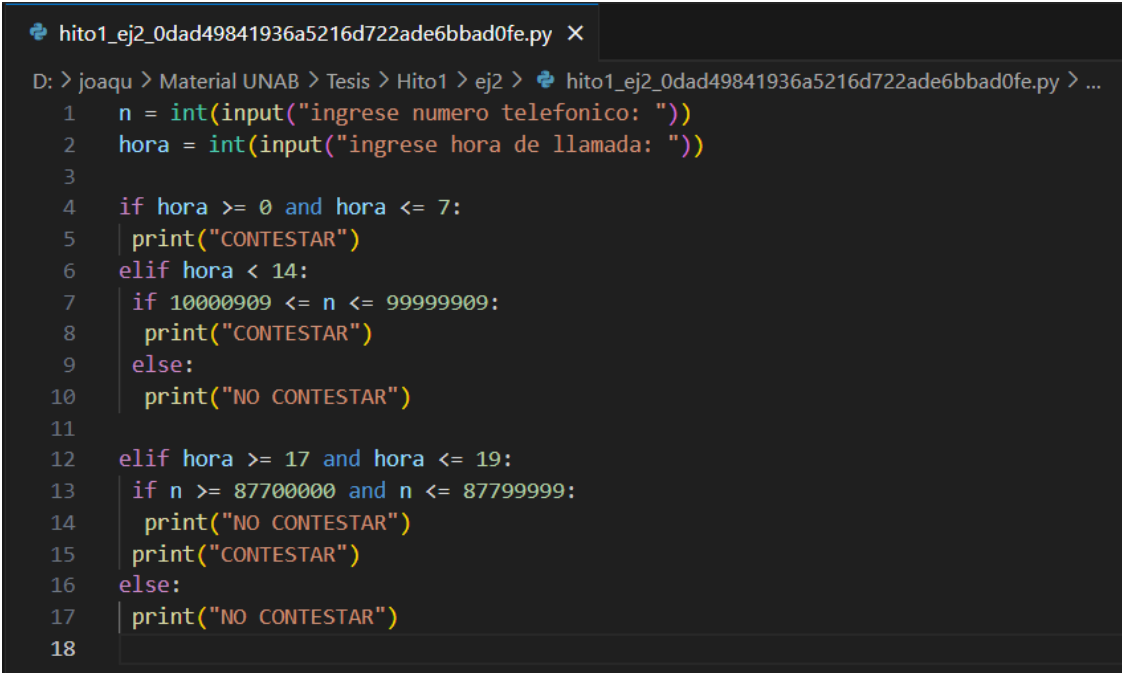
Figura 7.9: “Código Fuente Solución de Ejercicio 2 – Hito 1”



```
solucion.py X
D: > joaqu > Material UNAB > Tesis > Hito1 > ej2 > solucion.py > [?] n
1  n=input("Ingrese Numero Telefónico:")
2  m=int(input("Ingrese hora de la llamada:"))
3  if (0<=m<=7):
4      print("Contestar")
5
6  else:
7      if(7<m<=14):
8          o = n[5]
9          p = n[6]
10         q = n[7]
11         if (o=="9" and p=="0" and q=="9"):
12             print("Contestar")
13         else:
14             print("No Contestar")
15     else:
16
17         if (14<m<=17):
18             print("No Contestar")
19 if(17<m<=19):
20     r = n[0]
21     s = n[1]
22     t = n[2]
23     if(r=="8" and s=="7" and t=="7"):
24         print("No Contestar")
25     else:
26         print("Contestar")
27
28 elif (19<m<=23):
29     print("No Contestar")
30
```

La figura 7.9 enseña el script de la Solución del Ejercicio 2 para el Hito 1.

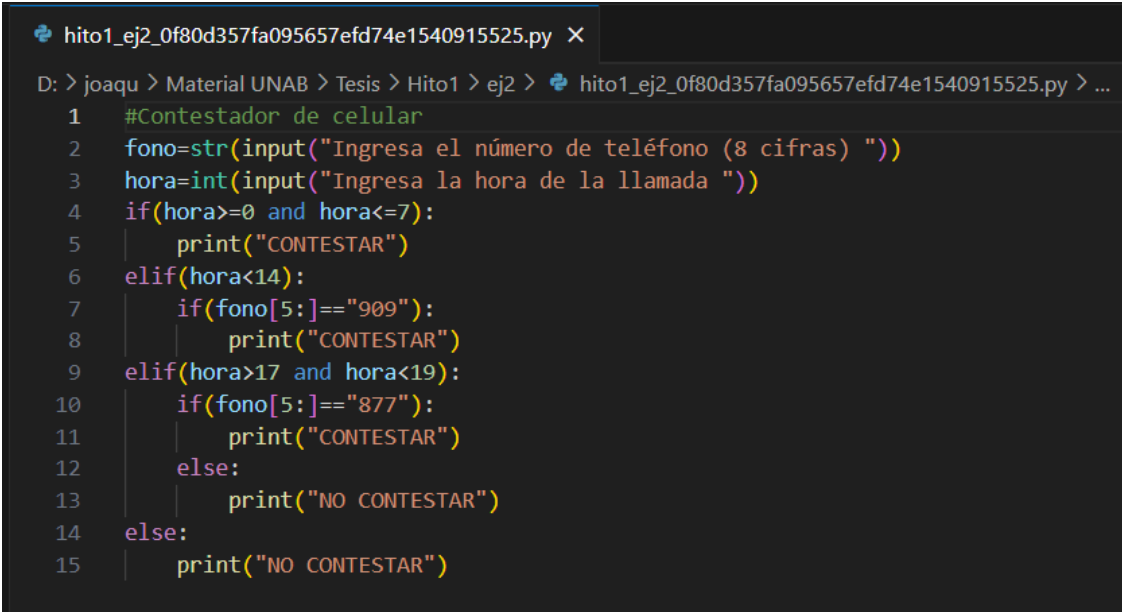
Figura 7.10: “Código Fuente Estudiante 1 de Ejercicio 2 – Hito 1”



```
hito1_ej2_0dad49841936a5216d722ade6bbad0fe.py X
D: > joaqu > Material UNAB > Tesis > Hito1 > ej2 > hito1_ej2_0dad49841936a5216d722ade6bbad0fe.py > ...
1  n = int(input("ingrese numero telefonico: "))
2  hora = int(input("ingrese hora de llamada: "))
3
4  if hora >= 0 and hora <= 7:
5      print("CONTESTAR")
6  elif hora < 14:
7      if 10000909 <= n <= 99999909:
8          print("CONTESTAR")
9      else:
10         print("NO CONTESTAR")
11
12 elif hora >= 17 and hora <= 19:
13     if n >= 87700000 and n <= 87799999:
14         print("NO CONTESTAR")
15         print("CONTESTAR")
16     else:
17         print("NO CONTESTAR")
18
```

La figura 7.10 corresponde al desarrollo entregado por el Estudiante 1 para el Ejercicio 2 del Hito 1.

Figura 7.11: “Código Fuente Estudiante 2 de Ejercicio 2 – Hito 1”



```
hito1_ej2_0f80d357fa095657efd74e1540915525.py X
D: > joaqu > Material UNAB > Tesis > Hito1 > ej2 > hito1_ej2_0f80d357fa095657efd74e1540915525.py > ...
1 #Contestador de celular
2 fono=str(input("Ingresa el número de teléfono (8 cifras) "))
3 hora=int(input("Ingresa la hora de la llamada "))
4 if(hora>=0 and hora<=7):
5     print("CONTESTAR")
6 elif(hora<14):
7     if(fono[5:]=="909"):
8         print("CONTESTAR")
9 elif(hora>17 and hora<19):
10    if(fono[5:]=="877"):
11        print("CONTESTAR")
12    else:
13        print("NO CONTESTAR")
14 else:
15    print("NO CONTESTAR")
```

La figura 7.11 muestra el desarrollo realizado por el Estudiante 2 para la entrega del Ejercicio 2 en el Hito 1.

Figura 7.12: “Código Fuente Solución de Ejercicio 3 – Hito 1”

```
solucion.py X
D: > joaqu > Material UNAB > Tesis > Hito1 > ej3 > solucion.py > ...
1  signos=[
2      ["aries",21,3,20,4],
3      ["tauro",21,4,21,5],
4      ["geminis",22,5,21,6],
5      ["cancer",22,6,22,7],
6      ["leo",23,7,22,8],
7      ["virgo",23,8,23,9],
8      ["libra",24,9,23,10],
9      ["escorpio",24,10,22,11],
10     ["sagitario",23,11,21,12],
11     ["capricornio",22,12,20,1],
12     ["acuuario",21,1,19,2],
13     ["piscis",20,2,20,3]
14 ]
15
16 def encontrar_signo(dia,mes):
17     for signo in signos:
18         if mes==signo[2] or mes==signo[4]:
19             if mes==signo[2] and dia>=signo[1]:
20                 return signo[0]
21             if mes==signo[4] and dia<=signo[3]:
22                 return signo[0]
23     return "error"
24
25 dia=int(input())
26 mes=int(input())
27 print(encontrar_signo(dia,mes))
```

La figura 7.12 enseña el script correspondiente a la Solución del Ejercicio 3 en el Hito 1.

Figura 7.13: “Código Fuente Estudiante 1 de Ejercicio 3 – Hito 1”

```
hito1_ej7_0dad49841936a5216d722ade6bbad0fe.py X
D: > joaqu > Material UNAB > Tesis > Hito1 > ej3 > hito1_ej7_0dad49841936a5216d722ade6bbad0fe.py > ...

1  #Zodiaco
2  dia = int(input("ingrese dia: "))
3  mes = int(input("ingrese mes: "))
4  if (dia >= 21 and mes == 3) or (dia <=20 and mes == 4) :
5      signo = ("aries")
6  if (dia >= 21 and mes == 4) or (dia <=21 and mes == 5):
7      signo = ("tauro")
8  if (dia >= 2 and mes == 5) or (dia <=21 and mes == 6):
9      signo = ("geminis")
10 if (dia >= 21 and mes == 6) or (dia <=23 and mes ==7):
11     signo = ("cancer")
12 if (dia >= 23 and mes == 7) or (dia <=23 and mes == 8):
13     signo = ("leo")
14 if (dia >= 23 and mes == 8) or (dia <=23 and mes == 9):
15     signo = ("virgo")
16 if (dia >= 23 and mes == 9) or (dia <=22 and mes == 10):
17     signo = ("libra")
18 if (dia >= 23 and mes == 10) or (dia <=22 and mes == 11):
19     signo = ("escorpio")
20 if (dia >= 23 and mes == 11) or (dia <=22 and mes == 12):
21     signo = ("sagitario")
22 if (dia >= 22 and mes == 12) or (dia <=20 and mes == 1):
23     signo = ("capricornio")
24 if (dia >= 20 and mes == 1) or (dia <=19 and mes == 2):
25     signo = ("acuuario")
26 if (dia >= 19 and mes == 2) or (dia <=21 and mes == 3):
27     signo = ("piscis")
28
29 print("Su signo es ",signo)
```

La figura 7.13 enseña el desarrollo hecho por el Estudiante 1 para la entrega del Ejercicio 3 en el Hito 1.

Figura 7.14: “Código Fuente Estudiante 2 de Ejercicio 3 – Hito 1”

```
hito1_ej7_0f80d357fa095657efd74e1540915525.py X
D: > joaqu > Material UNAB > Tesis > Hito1 > ej3 > hito1_ej7_0f80d357fa095657efd74e1540915525.py > ...

1  #Zodiaco
2  print("A partir de su día y mes de nacimiento le indicaremos su signo zodiacal")
3  dia=int(input("Ingrese su día de nacimiento "))
4  mes=int(input("Ingrese su mes de nacimiento "))
5  if(mes==1):
6      if(dia<=20):
7          print("Eres Capricornio")
8      else:
9          print("Eres Acuario")
10 if(mes==2):
11     if(dia<=19):
12         print("Eres Acuario")
13     else:
14         print("Eres Piscis")
15 if(mes==3):
16     if(dia<=21):
17         print("Eres Piscis")
18     else:
19         print("Eres Aries")
20 if(mes==4):
21     if(dia<=20):
22         print("Eres Aries")
23     else:
24         print("Eres Tauro")
25 if(mes==5):
26     if(dia<=21):
27         print("Eres Tauro")
28     else:
29         print("Eres Géminis")
30 if(mes==6):
31     if(dia<=21):
32         print("Eres Géminis")
33     else:
34         print("Eres Cáncer")
35 if(mes==7):
36     if(dia<=21):
37         print("Eres Cáncer")
38     else:
39         print("Eres Leo")
40 if(mes==8):
41     if(dia<=23):
42         print("Eres Leo")
43     else:
44         print("Eres Virgo")
45 if(mes==9):
46     if(dia<=23):
47         print("Eres Virgo")
48     else:
```

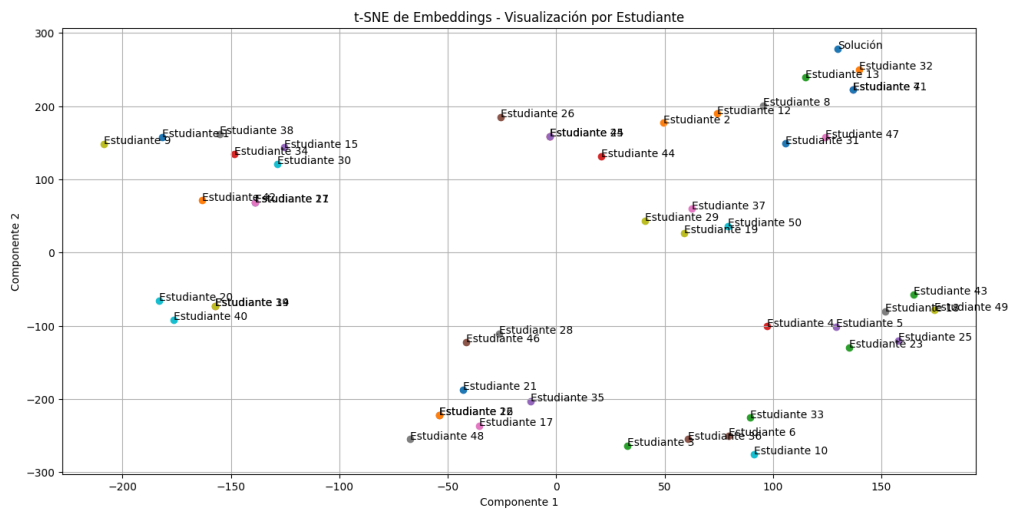
La figura 7.14 muestra la mayor parte del script desarrollado por el Estudiante 2 para la entrega del Ejercicio 3 en el Hito 1.



- Análisis Estudiantes: Hito 2

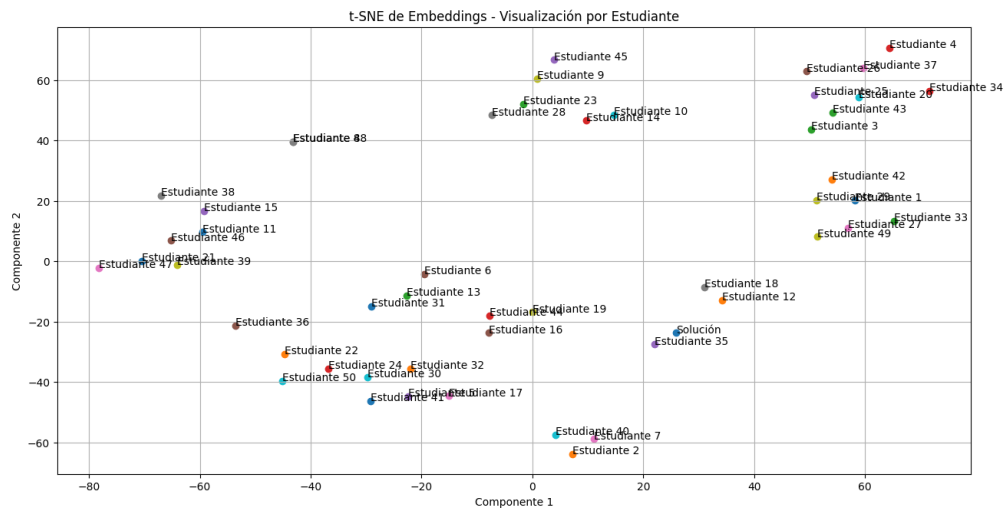
Para el Hito 2, al igual que en Hito 1, se generan embeddings a partir de los códigos de estudiantes en sus respectivos ejercicios. En las figuras 4, 5 y 6 podemos visualizar los resultados de las agrupaciones de vectores en sus espacios bidimensionales usando los métodos t-SNE y PCA nuevamente.

Figura 7.15: “Gráfico t-SNE de Ejercicio 1 – Hito 2”



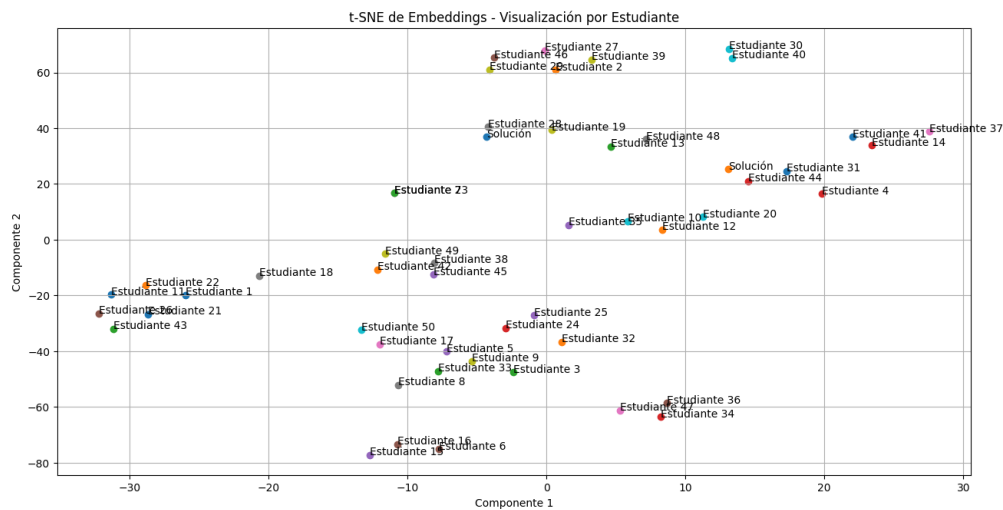
La figura 7.15 enseña el gráfico realizado a través de los datos del Ejercicio 1 del Hito 2, en donde se aprecia una dispersión de datos evidente con diversas agrupaciones de estudiantes moderadamente alejadas del Punto Solución.

Figura 7.16: “Gráfico t-SNE de Ejercicio 2 – Hito 2”



La figura 7.16 presenta el grafico creado con los archivos de los estudiantes entregados para el Ejercicio 2 en el Hito 2, donde ya no existen diversos grupos y en vez se presentan 2 grandes agrupaciones, cerca y lejos de Punto Solución, el cual se encuentra entre estos dos grupos, con algunas excepciones de estudiantes entre estos grupos tanto cerca como lejos del Punto Solución.

Figura 7.17: “Gráfico t-SNE de Ejercicio 3 – Hito 2”



La figura 7.17 muestra el grafico realizado con los datos de los estudiantes en el Ejercicio 3 del Hito 2, donde la dispersión de datos es moderada con pocas agrupaciones específicas, pero con variadas ramificaciones alrededor el Punto Solución.

- Análisis Estudiantes: Hito 2

Los procesos de análisis en Hito 2 son los mismos que en Hito 1, nuevamente se eligen a 2 estudiantes que estén presentes en cada ejercicio, de entre una muestra de los primeros 50 dentro de cada Ejercicio en el Hito 2. Para el Estudiante 1 (3a404eb99342dcad9b2f23e1e08ec05c), nuevamente buscando su ID en los ejercicios dentro de Hito 2, para los ejercicios 1, 2 y 3, el script lo identifica como “Estudiante 13” para Ejercicio 1, “Estudiante 30” para Ejercicio 2 y “Estudiante 21” para Ejercicio 3.

Finalmente, para el Estudiante 2 (b497197d056a2381f0ace6e93017597f), se busca su ID en los ejercicios 1, 2 y 3 del Hito 2 en los que ese se encuentra, el script lo identifica como “Estudiante 49” para los Ejercicios 1, 2 y 3 debido a su posición dentro de las 50 entradas de estudiantes en la muestra elegida para analizar Hito 1.

Figura 7.18: “Código Fuente Solución de Ejercicio 1 – Hito 2”

```
solucion.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej1 > solucion.py > ...
1  s1="ACCTGGTTCTGTAGTCAGGATTACTA"
2  s2="TGACGTTTCAGTAGTCGATT"
3  def alinear(s1,s2):
4      salida=[]
5      s1=list(s1)
6      s2=list(s2)
7      j=0
8      i=0
9      coincidencia=False
10     while i<len(s1):
11         if s1[i]!=s2[j]:
12             s2.insert(j,"_")
13             i=i+1
14             j=j+1
15         else:
16             coincidencia=True
17             i=i+1
18             j=j+1
19         if j>=len(s2):
20             break
21     return "".join(s2)
22
23 s1=input("s1: ")
24 s2=input("s2: ")
25 r2=alinear(s1,s2)
26 print(r2)
```

La figura 7.18 muestra el script correspondiente a la Solución del Ejercicio 1 del Hito 2.

Figura 7.19: “Código Fuente Estudiante 1 de Ejercicio 1 – Hito 2”

```
hito2_ej1_3a404eb99342dcad9b2f23e1e08ec05c.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej1 > hito2_ej1_3a404eb99342dcad9b2f23e1e08ec05c.py > ...
1  def secuencias(secuencia1,secuencia2):
2      x=0
3      y=0
4      lista2=[]
5      while x<len(secuencia1) and y<len(secuencia1):
6          letra2=secuencia2[y]
7          letra1=secuencia1[x]
8          if letra1==letra2:
9              lista2.append(letra2)
10             x=x+1
11             y=y+1
12         elif letra1!=letra2:
13             lista2.append("_")
14             x=x+1
15             y=y
16
17     for x in range(y,len(secuencia2)):
18         posicion2=secuencia2[x]
19         lista2.append(posicion2)
20
21     final="".join(lista2)
22     print(final)
23
24     secuencia1=input("ingrese secuencia 1: ")
25     secuencia2=input("ingrese secuencia 2: ")
26     secuencias(secuencia1,secuencia2)
27
```

La figura 7.19 corresponde al script desarrollado realizado por el Estudiante 1 para la entrega del Ejercicio 1 en el Hito 2.

Figura 7.20: “Código Fuente Estudiante 2 de Ejercicio 1 – Hito 2”

```
hito2_ej1_b497197d056a2381f0ace6e93017597f.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej1 > hito2_ej1_b497197d056a2381f0ace6e93017597f.py > ...
1  modelo = str(input())
2  fix = str(input())
3
4  i = 0
5  while i <= len(fix):
6      fix = list(fix)
7      nro_guiones = modelo.find(fix[i], i) - i
8      fix.insert(i, " " * nro_guiones)
9      fix = "".join(fix)
10     i += (1 + abs(nro_guiones))
11 print(fix)
12
```

La figura 7.20 muestra el script desarrollado por el estudiante 2 para la entrega del Ejercicio 1 del Hito 2.

Figura 7.21: “Código Fuente Solución de Ejercicio 2 – Hito 2”

```
solucion.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej2 > solucion.py > ...
1  def validar_secuencia(s):
2      for letra in s:
3          if letra.upper() not in ["A", "C", "T", "G"]:
4              return False
5      return True
6
7  secuencia=input("Ingrese secuencia: ")
8  if validar_secuencia(secuencia):
9      print("La secuencia es correcta")
10 else:
11     print("La secuencia es incorrecta")
12
13
```

La figura 7.21 muestra el script perteneciente a la Solución del Ejercicio 2 del Hito 2.

Figura 7.22: “Código Fuente Estudiante 1 de Ejercicio 2 – Hito 2”

```
hito2_ej2_3a404eb99342dcad9b2f23e1e08ec05c.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej2 > hito2_ej2_3a404eb99342dcad9b2f23e1e08ec05c.py > ...
1  sec=input("ingrese secuencia de ADN: ")
2  sec=sec.upper()
3  lista=list(sec)
4  f=0
5  for x in range(0,len(sec)):
6      d=lista[x]
7      if d!="G" and d!="A" and d!="T" and d!="C":
8          print("secuencia incorrecta")
9          f=1
10         break
11 if f==0:
12     print("secuencia correcta")
13
14
```

La figura 7.22 corresponde al desarrollo realizado por el Estudiante 1 para la entrega del Ejercicio 2 del Hito 2.

Figura 7.23: “Código Fuente Estudiante 2 de Ejercicio 2 – Hito 2”

```
hito2_ej2_b497197d056a2381f0ace6e93017597f.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej2 > hito2_ej2_b497197d056a2381f0ace6e93017597f.py > ...
1  letras = "ACTG"
2  sec = input()
3  win = True
4  for a in sec:
5      if a.capitalize() not in sec:
6          print("secuencia incorrecta")
7          win = False
8          break
9
10 if win:
11     print("secuencia correcta")
12
```

La figura 7.23 corresponde al desarrollo del Estudiante 2 para la entrega del Ejercicio 2 del Hito 2.

Figura 7.24: “Código Fuente Solución de Ejercicio 3 – Hito 2”

```
solucion.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej3 > solucion.py > ...
1  print('** SUBSECUENCIAS DE ADN **')
2
3  def sub_adn(sequencia,numero): # numero = 3 lista_sec = ['A','G','C','G'] len = 4
4      sequencia = sequencia.upper()
5      lista_sec = list(sequencia)
6      lista_gen = []
7      lista = []
8      p = 0
9      while p < len(lista_sec): #Si es que p = 4 (len), se quiebra el ciclo
10         lista.append(lista_sec[p]) #el p -= (numero-1) no tiene efecto en este caso
11         p += 1
12         if len(lista)%numero == 0:
13             lista_gen.append(lista)
14             lista = [] #reaunudamos la lista
15             p -= (numero-1) #ejem: 3-2=1
16     lista_gen1 = []
17     for i in lista_gen:
18         lista_gen1.append(''.join(i))
19     resultado=[]
20     for i in lista_gen1:
21         if lista_gen1.count(i) == 1:
22             resultado.append(i)
23     #las funciones debieran retornar su resultado con return y no usar print
24     return resultado
25
26     sequencia = input('Ingrese la secuencia: ')
27     numero = int(input('Número: '))
28     resultado=sub_adn(sequencia,numero)
29     #cuando llamas la función guardas su resultado en un variable
30     #luego analizar el resultado y de acuerdo a su valor el programa continúa
31     if len(resultado)==0:
32         print("ninguna")
33     else:
34         print(resultado)
35
```

La figura 7.24 muestra el script de una de las posibles soluciones para el Ejercicio 3 en el Hito 2.



Figura 7.25: “Código Fuente Solución Base de Ejercicio 3 – Hito 2”

```
solucion_base.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej3 > solucion_base.py > ...
1  s=input("Inserte Secuencia de ADN: ").upper()
2  n=int(input("Inserte tamaño de la subsecuencia a buscar:" ))
3  l1=[]
4  for i in range(0,len(s)-n+1):
5      s1=s[i:i+n]
6      for j in range(0,len(s)-n+1):
7          if j!=i:
8              s2=s[j:j+n]
9          else:
10             continue
11         if s1==s2:
12             break
13         else:
14             if j==len(s)-n:
15                 l1.append(s1)
16  if len(l1)==0:
17      print("ninguna")
18  else:
19      for secuencia in l1:
20          print(secuencia)
21
```

La figura 7.25 muestra el segundo script posible para la Solución del Ejercicio 3 del Hito 2.

Figura 7.26: “Código Fuente Estudiante 1 de Ejercicio 3 – Hito 2”



```
hito2_ej3_3a404eb99342dcad9b2f23e1e08ec05c.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej3 > hito2_ej3_3a404eb99342dcad9b2f23e1e08ec05c.py > ...
1  a=input("")
2  n=int(input(""))
3  lista=[]
4  listam=[]
5  for b in range(n,len(a)+1):
6      g=a[b-n:b]
7      if(g in lista):
8          lista.remove(g)
9          listam.append(g)
10     if (g not in listam):
11         lista.append(g)
12 if(lista!=[]):
13     for a in lista:
14         print(a)
15 else:
16     print("ninguna")
17
```

La figura 7.26 corresponde al desarrollo realizado por el Estudiante 1 para la entrega del Ejercicio 3 del Hito 2.

Figura 7.27: “Código Fuente Estudiante 2 de Ejercicio 3 – Hito 2”

```
hito2_ej3_b497197d056a2381f0ace6e93017597f.py X
D: > joaqu > Material UNAB > Tesis > Hito2 > ej3 > hito2_ej3_b497197d056a2381f0ace6e93017597f.py > ...
1  sec = str(input())
2  largo = int(input())
3
4  i = 0
5  ninguna = True
6  while i <= len(sec) - largo:
7      copia = list(sec)
8      test = sec[i : i + largo]
9      print("test = ", test)
10     if i == 0:
11         copia = "".join(copia[i + largo :])
12     else:
13         copia = "".join(copia[0 : i] + copia[i + largo :])
14     print("copia", copia)
15     if copia.count(test) == 0 and sec != "AAAAA":
16         ninguna = False
17         print (test)
18     i = i + 1
19
20 if ninguna:
21     print("ninguna")
```

La figura 7.27 presenta el desarrollo realizado por el Estudiante 2 para el Ejercicio 3 del Hito 2.

## 8. Discusión.

Según los análisis de los resultados para cada Hito, dentro del Hito 1, El Estudiante 1 (archivo 0dad49841936a5216d722ade6bbad0fe) dentro del Ejercicio 1 existe un respeto hacia la estructura general que se esperaba obtener para indicar un buen desempeño, comparando con el desarrollo del archivo Solución, utiliza nombres de variables diferentes, utiliza un método distinto para imprimir el valor final al método utilizado en la Solución, pero mantiene el mismo flujo principal. En el Ejercicio 2, existe una omisión en el manejo del rango de horario, cae en errores lógicos como imprimir una respuesta incorrecta en ciertos casos a pesar de tener una estructura más legible, y no mantiene un criterio en la revisión de dígitos para el número telefónico. Para el Ejercicio 3,

utiliza un método directo de *if* sueltos para revisar los signos zodiacales, no utiliza la estructura de código esperada como una lista con los signos dentro ni una función de búsqueda, lo cual se conoce como solución “fuerza bruta”, teniendo una logística simple y propensa a errores.

Para el análisis del Estudiante 2 (archivo 0f80d357fa095657efd74e1540915525) del Hito 1, en el Ejercicio 1 utiliza variables similares, pero cae en un error notable repitiendo el mismo input para todas las notas en vez de indicarlo de forma específica para cada una, a pesar de eso realiza cálculos correctos y mantiene una organización simple. Para el Ejercicio 2, obtiene correctamente el número telefónico como *string* y los revisa también de manera correcta, sin embargo, existen errores lógicos en la revisión horaria, no termina el desarrollo de la decisión del rango de horas, omitiendo los mismos valores que el Estudiante 1, y teniendo chequeos mal estructurados. Para el Ejercicio 3, implementa una estructura similar de “fuerza bruta” pero extensa, respeta de manera correcta las diferencias de signos según sus respectivas fechas, pero no utiliza ninguna lista, pero a pesar de sus fallos su implementación es clara.

Analizando el Hito 2, el Estudiante 1 (archivo 3a404eb99342dcad9b2f23e1e08ec05c), en el Ejercicio 1 utiliza una función diferente en su estructura respecto a lo esperado en la solución oficial, maneja de forma correcta el desplazamiento de las secuencias pero el trabajo de cada indica puede generar inconsistencias cuando las entradas son de diferente longitud, utiliza caracteres de la segunda secuencia de forma explícita, lo que genera errores nuevamente al tener entradas de diferentes longitudes, esto provoca que su respuesta sea una estructura clara y generalmente estable, pero no manejable. Para el Ejercicio 2 obtiene los caracteres de la secuencia correctamente, utiliza un recorrido de datos tradicional para analizar la entrada de datos y su verificación es correcta, pero toda su estructura es extensa a diferencia de la Solución esperada donde todo se condensa en una función. Para el Ejercicio 3 la lógica entregada por el Estudiante es similar a lo esperado, a diferencia de ambas soluciones, el estudiante realiza su lógica en un solo bloque de código que cumple con el objetivo del ejercicio, haciendo que su lógica funcione de forma forzada y clara pero no de forma modular u óptima.

Finalmente, el Estudiante 2 (archivo b497197d056a2381f0ace6e93017597f) del Hito 2, en el Ejercicio 1 implementa una solución propia, pero más enredada que la solución original esperada lo cual presenta riesgo de errores debido a las manipulaciones dentro de su propio bucle *while* y tampoco se controla explícitamente los límites de índices. En el Ejercicio 2 captura la secuencia

correctamente, pero tiene errores condicionales al comparar las mismas secuencias ingresadas en vez de las necesarias según la Solución esperada, esto deriva en que su validación es lógica pero errónea en la manera de ejecutarla. En el Ejercicio 3, realiza validaciones en cada subsecuencia fuera del rango inicial, tiene fallos como copiar cada iteración del mensaje impreso, generando ruido innecesario, a pesar de eso la condición final para imprimir “ninguna” es correcta, esto lleva a tener una lógica detallada pero desorganizada comparándola con la Solución esperada.

## 9. Conclusiones

En relación con los objetivos del proyecto, se logró evaluar la factibilidad de utilizar vectorizaciones generadas por modelos pre entrenados para representar códigos fuentes entregados por estudiantes de cursos de programación. A través del uso del modelo CodeBERT y de técnicas de reducción dimensional como t-SNE. También considerando los resultados obtenidos en ambos Hitos elegidos y analizados, se consiguió visualizar diferencias estructurales y semánticas en los códigos obtenidos, cumpliendo así el objetivo general de representar y analizar el desempeño estudiantil de manera significativa.

En cuanto al primer objetivo específico, se construyó un dataset con códigos fuente y sus respectivas soluciones segmentadas por Ejercicios e Hitos. Si bien no fue posible incluir notas finales, se priorizo la opción de seleccionar casos representativos a lo largo de Hitos y Ejercicios.

Respecto al segundo objetivo específico, se consiguió utilizar CodeBERT como modelo base para generar vectorizaciones y analizar su comportamiento frente a las diferentes entregas. A pesar de no realizar una comparación formal con otros modelos, los resultados con CodeBERT permitieron verificar que esta técnica puede capturar diferencias relevantes entre soluciones y entregas de estudiantes.

Para el Hito 1, el Estudiante 1 se refleja en los gráficos de cada Ejercicio siendo visible como en el gráfico 1 el Estudiante 1 (como Estudiante 26 bajo la muestra de datos) tiene su respuesta bastante cerca del punto Solución, mientras que en el resto de los gráficos para los Ejercicios 2 y 3 (siendo enumerado como Estudiante 22 para ambos gráficos), este se aleja de manera consistente,

mostrando una evidencia de que el modelo CodeBERT comprende que según los componentes de sus respuestas.

A comparación del Estudiante 1, el análisis del Estudiante 2 indica cierto nivel de mejora a medida que desarrolla los ejercicios, comenzando con que en el grafico del Ejercicio 1, el Estudiante 2 (como Estudiante 44) tiene su punto notablemente alejado del punto Solución siendo incluso uno de los dos puntos que más se alejan de la gráfica. Para el Ejercicio 2, su punto ya no se encuentra notablemente alejado del punto Solución, pero aun así se encuentra junto a la agrupación más alejada de la Solución, lo cual demuestra que cometer los mismos errores que el Estudiante 1 se demuestra en la gráfica de sus desarrollos. Para el Ejercicio 3, a pesar de que su manera de estructurar el desarrollo esperado es extensamente diferente a la Solución esperada, el hecho de haber conseguido un desarrollo claro hace que se encuentre dentro de la agrupación que rodea de forma cercana al punto Solución.

Para las conclusiones del Estudiante 1 del Hito 2, se representa en los gráficos como se visualiza un decaimiento en su manera de desarrollar código al tener cada punto de sus respuestas alejándose cada vez mas de cada punto Solución, el cuarto grafico (en donde se ve como Estudiante 13) estando dentro de los puntos más cercanos a la Solución esperada, en el quinto grafico (visto como Estudiante 30) estando alrededor de media distancia del punto Solución en relación a los estudiantes más alejados, y en el grafico seis (estando como Estudiante 21) llegando estar dentro de la agrupación más alejada incluso habiendo dos soluciones posibles para dicho ejercicio.

Y, por último, para el Estudiante 2 (estando en todos los gráficos como Estudiante 49) se visualiza como se mantiene su distancia a lo largo de los ejercicios del Hito 2. Dentro del cuarto grafico se encuentra a una distancia media al punto Solución junto a varios Estudiantes, en el quinto grafico acercándose un poco estando en una de las agrupaciones que se acercan a la Solución esperada, y finalmente para el sexto grafico encontrando al estudiante nuevamente a una distancia medianamente alejada, lo cual se comprende según la visualización de sus múltiples desarrollos en donde mantiene aspectos tanto correctos como incorrectos.

En resumen, el análisis visual mediante embeddings y reducción de dimensionalidad demostró ser una herramienta válida para identificar patrones de desempeño en estudiantes de programación con los datos utilizados. El uso de CodeBERT como modelo base permitió obtener las diferencias

semánticas relevantes necesarias para validar la hipótesis de que los modelos basados en la arquitectura transformers pueden ser útiles dentro del contexto educativo para apoyar la evaluación del aprendizaje dentro de los cursos de programación.

## **10. Limitaciones del trabajo y trabajos futuros**

Se considera el desarrollo del proyecto y sus análisis con solo un modelo pre entrenado, lo cual representa la principal limitación de este aun habiendo conseguido representar la vectorización de los códigos como se había propuesto. Y aunque los resultados fueron satisfactorios, la mayor recomendación de continuaciones de estudio es ampliar la comparación incorporando otros modelos basados en la arquitectura Transformers para evaluar si algunos o la mayoría ofrecen mejoras en la obtención de estructuras y semánticas, o en su defecto presentan peores resultados que el actual proyecto.

Una segunda posible limitación es la ausencia de comparación respecto a las calificaciones finales de los estudiantes analizados, lo cual impide establecer correlaciones directas, pero esto mismo puede realizarse al finalizar de las comparaciones con los otros modelos pre entrenados como se sugirió recientemente y sumarlo para visualizar una comparativa completa entre todos los resultados de estudios de los modelos pre entrenados con las calificaciones finales y establecer un punto generalizado hacia el tipo de arquitectura Transformers y no junto a los modelos en sí.

Así mismo, el tamaño del dataset, siendo dos Hitos con tres Ejercicios cada uno, con un alrededor de 50 entregas de estudiantes en cada uno, a pesar de ser bastante representativo al común en un curso de programación en la educación superior, podría ampliarse aún más abarcando más cursos, diferentes niveles de dificultad e incluso mayor diversidad de estudiantes.

- Agregar el uso de vectorización para correlacionar con el rendimiento académico

Como líneas directas de trabajos futuros se sugiere:

- Aplicar misma metodología a datasets similares, pero en otros lenguajes, principalmente C++, pero también aplicables a Java o JavaScript.

- Implementar predicciones de rendimiento basados en los embeddings desarrollados que permitan anticipar posibles riesgos de reprobación o incluso una necesidad de apoyo adicional.
- Integrar visualizaciones dinámicas en plataformas educativas, considerando incluso un monitoreo en tiempo real del progreso de los estudiantes.

Dichas extensiones pueden permitir la utilidad de los modelos de vectorización de código basados en la arquitectura Transformers como herramientas de apoyo en la docencia de cursos de programación en la educación superior.

## 11. Referencias

1. Zhang, Y. Chen, S. Oney (2023), Identifying misunderstandings by visualizing students' Coding Process. Recuperado el 7 de octubre de 2023 de:  
  
[VizProg: Identifying Misunderstandings By Visualizing Students' Coding Progress | Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems](#)
2. Azcona, P. Arora, I.-H. Hsiao, A. Smeaton (2019), user2code2vec: Embeddings for profiling students based on distributional representations of source code, Recuperado el 7 de octubre de 2023 de:  
  
[user2code2vec | Proceedings of the 9th International Conference on Learning Analytics & Knowledge](#)
3. Rakhmanov (2020), A comparative study on vectorization and classification techniques in sentiment analysis to classify student-lecturer Comments, Recuperado el 7 de octubre de 2023 de:  
  
[A Comparative Study on Vectorization and Classification Techniques in Sentiment Analysis to Classify Student-Lecturer Comments - ScienceDirect](#)