



UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY



FACULTAD DE
INGENIERÍA

Corrección automática de ejercicios de inglés utilizando modelos neuronales

Informe de Proyecto de Grado presentado por

Martín Manitto y ZuoHeng Dai

en cumplimiento parcial de los requerimientos para la graduación de la carrera
de Ingeniería en Computación de Facultad de Ingeniería de la Universidad de
la República

Supervisores

Aiala Rosá
Luis Chiruzzo

Montevideo, 27 de diciembre de 2024



Corrección automática de ejercicios de inglés utilizando modelos neuronales por Martín Manitto y ZuoHeng Dai tiene licencia [CC Atribución 4.0](https://creativecommons.org/licenses/by/4.0/).

Agradecimientos

Queremos expresar nuestro más sincero agradecimiento a nuestros tutores, quienes siempre estuvieron dispuestos a ofrecernos recomendaciones valiosas sobre artículos y temas relacionados con nuestra investigación. Su orientación y sugerencias de alternativas para resolver los desafíos que enfrentamos han sido invaluable para el progreso de este proyecto.

Además, queremos agradecer también al Centro Nacional de Supercomputación(ClusterUY) por brindarnos acceso a sus recursos computacionales durante el desarrollo de este trabajo. El acceso a sus computadoras de alta capacidad,especialmente aquellas equipadas con GPUs, fue fundamental para poder ejecutar modelos como LLaMA y Mistral 7B.

Resumen

En los últimos años, ha crecido la importancia de la enseñanza del inglés en Uruguay. Sin embargo, el país enfrenta un desafío significativo: la escasez de docentes, lo que dificulta que los estudiantes alcancen un nivel adecuado en su educación. Este problema afecta especialmente a los niños en los niveles iniciales, limitando su progreso en el aprendizaje del idioma.

Este proyecto continúa el trabajo de iniciativas anteriores que buscan apoyar a los docentes en la corrección de ejercicios escritos por estudiantes. El propósito central del proyecto es desarrollar una herramienta que facilite la corrección automática de textos escritos por estudiantes de inglés en etapas iniciales. La herramienta debe recibir los textos y devolver una versión corregida, señalando los errores identificados.

En un inicio, exploramos el uso exclusivo de Grandes Modelos de Lenguaje (LLMs por su sigla en inglés), para abordar este problema. Sin embargo, tras varios experimentos, los resultados no fueron satisfactorios. Ante esta limitación, optamos por una solución alternativa que combina el uso de LLMs, una herramienta para marcar diferencias y un clasificador entrenado para predecir los tipos de errores, logrando el mismo objetivo de corrección.

El sistema desarrollado se compone de tres módulos independientes que pueden funcionar por separado o de manera conjunta. Obtuvimos un corrector basado en Mistral, con una puntuación $F_{0,5}$ de 0.80, y un clasificador de errores con un F_1 de 0.76. En conjunto, el sistema alcanzó un rendimiento global de $F_{0,5}$ de 0.60.

Índice general

1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Objetivo general	2
1.3. Objetivos específicos	2
1.4. Estructura del documento	2
2. Revisión de antecedentes	5
2.1. Competencias anteriores	5
2.1.1. CoNLL-2013	6
2.1.2. CoNLL-2014	7
2.1.3. BEA-2019	9
2.1.4. MultiGED-2023	10
2.1.5. Proyecto de Grado: Construcción de herramientas de soporte para corrección en enseñanza de inglés	11
2.2. Corpus	12
2.2.1. Corpus disponibles	12
2.2.2. Corpus utilizado para el proyecto	13
2.3. Herramientas utilizadas	16
2.3.1. Aprendizaje automático supervisado	17
2.3.2. Modelos de lenguaje	18
2.3.3. Métricas	21
3. Descripción de la solución	25
3.1. Descripción de la problemática	25
3.2. Propuesta de solución	26
3.3. Primer enfoque con LLMs	26
3.3.1. Examinar la mejor versión de prompt en ChatGPT	26
3.4. Solución final desarrollada	27
3.5. Textos etiquetados	28
3.6. Preprocesamiento de textos	28
3.6.1. Selección y edición de textos	29
3.6.2. Preprocesamiento del texto	30
3.7. Corrección de textos utilizando LLMs	31
3.7.1. Elección y ejecución de LLMs	31

3.7.2.	Corrección de textos	32
3.8.	Marcado de diferencias utilizando WER	32
3.8.1.	Alineado de textos	33
3.8.2.	Insertado de tags	34
3.9.	Clasificador de errores	36
3.9.1.	Selección de características para el clasificador	37
3.9.2.	Entrenamiento del clasificador	38
4.	Experimentación	41
4.1.	Métricas de corrección para el corpus del proyecto 2021 usando ChatGPT y LLaMA	41
4.2.	Pruebas realizadas con ChatGPT y LLaMA	43
4.3.	Pruebas realizadas con nuestra solución	47
4.3.1.	Pruebas del clasificador aislado	47
4.3.2.	Pruebas de la solución completa	49
5.	Conclusiones y Trabajo Futuro	53
5.1.	Resultados finales	53
5.2.	Dificultades encontradas	54
5.2.1.	Falta de conocimiento previo	54
5.2.2.	Limitaciones técnicas en la ejecución de modelos de lenguaje	55
5.2.3.	Desafíos derivados del corpus	55
5.3.	Trabajo futuro	55
5.3.1.	Corrección de textos	56
5.3.2.	Marcado de textos corregidos	56
5.3.3.	Clasificación de errores	56
5.3.4.	Consideración sobre los tipos de correctores	57
5.3.5.	Creación de una aplicación final	57
	Referencias	63
A.	Anexo 1	67
A.1.	Algoritmo de Distancia de Levenshtein	67
A.2.	Prompts	67
A.2.1.	Prompt 1	67
A.2.2.	Prompt 2	68
A.2.3.	Prompt 3	69
A.2.4.	Prompt 4	70
A.2.5.	Prompt Version Final	70
A.2.6.	Prompt usado para Mistral	71
A.3.	Textos usados para entrenar al modelo	71

Capítulo 1

Introducción

En Uruguay, desde hace ya algunos años, se le da cada vez más importancia tanto a la enseñanza de inglés como al uso de herramientas informáticas en la educación. Hace dos décadas no todas las escuelas disponían de profesor de inglés o de recursos informáticos, pero, a través de programas como el Plan CEIBAL, y proyectos como Uruguay Plurilingüe 2030¹, se ha alcanzado la universalidad de la enseñanza de este idioma a nivel de escuelas urbanas.

Sin embargo, aún se requiere más trabajo para alcanzar a los estudiantes de escuelas rurales, donde se dispone de menor cantidad de recursos sobre todo humanos. Es decir, en las escuelas rurales no siempre se dispone de un maestro por grado, y mucho menos de profesores de inglés suficientes para dichas escuelas. A esto se le suma una menor tasa de accesibilidad a internet, lo cual impide que se lleven a cabo regularmente clases virtuales con profesores a distancia.

Por ello, para disminuir parcialmente esa brecha, se busca a través del uso de herramientas informáticas poder suplir algunas de las tareas típicamente llevadas a cabo por profesores. En nuestro caso, buscamos generar una herramienta que pudiera identificar y corregir errores en textos escritos por alumnos de inglés, a modo de simplificar o reducir la tarea de los profesores.

Para llevar a cabo esta tarea, nos basamos en trabajos surgidos en competencias relacionadas con el área de PLN² y en particular con la tarea de corrección de textos. Disponemos a su vez de un corpus de textos escritos por alumnos de escuelas primarias de todo el país, algunos de los cuales ya están corregidos a mano dado que el mismo corpus se utilizó en un trabajo anterior relacionado con esta misma tarea.

1.1. Motivación del proyecto

Se busca reducir el trabajo requerido por los profesores a la hora de corregir textos escritos por estos estudiantes, para que la menor cantidad de docentes

¹<https://www.anep.edu.uy/codicen/politicas-linguisticas/focus-on-first>

²Procesamiento de Lenguaje Natural

disponibles para ellos no sea una limitante tan grande en la enseñanza del idioma inglés, y de esta forma reducir la distancia entre ellos y los estudiantes en zonas más privilegiadas en este sentido.

Por nuestra parte, nos interesa continuar aprendiendo sobre técnicas más modernas en el área de PLN, a las que no se les dedica tanto tiempo en los cursos previos que tomamos en dicha área. También consideramos que es positivo investigar y conocer más sobre LLMs ya que es un área en pleno desarrollo en este momento, con herramientas nuevas que van apareciendo día a día.

1.2. Objetivo general

El objetivo general es crear una herramienta para corregir textos en inglés. Esta herramienta debe recibir textos escritos por los niños que posiblemente contengan errores, y devolver sugerencias pertinentes para su corrección.

1.3. Objetivos específicos

- Análisis de los diferentes enfoques para realizar correcciones de textos a través de la lectura de *papers*, especialmente de los surgidos en competencias con objetivos similares al nuestro, enfatizando aquellas técnicas surgidas más recientemente y en particular el uso de LLMs.
- Relevamiento y análisis de diferentes LLMs para su aplicación en la tarea planteada, especialmente de modelos de uso libre.
- Experimentación con técnicas de *prompting* para corrección de textos.
- Creación de una herramienta que utilice estos modelos. La herramienta debe recibir como entrada textos a corregir y retornar como salida los mismos textos corregidos y los errores identificados.

1.4. Estructura del documento

A continuación se presenta una descripción general de la estructura de los capítulos siguientes:

En el capítulo 2, se realiza un estado del arte sobre este tema. Se listan las competencias relacionadas con el proyecto, los corpus utilizados y se describe un trabajo anterior relacionado con nuestro proyecto. Se detallan en subsecciones las herramientas disponibles y el conocimiento relacionado con temas como la arquitectura de transformers, modelos de lenguaje y métricas de distancia entre textos.

En el capítulo 3, se describe el procedimiento seguido y las decisiones tomadas durante el desarrollo del proyecto. Se describe la solución implementada y por qué se realizó de esta forma.

En el capítulo 4, se presenta la experimentación realizada, mostrando los resultados de ejecutar la solución con dos corpus distintos. También mostramos los resultados de experimentos previos que llevaron a la implementación de esta solución.

En el capítulo 5, se exponen las conclusiones obtenidas del estudio realizado, algunas dificultades que tuvimos en el proyecto y las áreas que podrían ser objeto de futuras investigaciones o mejoras.

Capítulo 2

Revisión de antecedentes

El problema abordado en este proyecto se encuentra dentro del campo de *Grammatical Error Detection (GED)*¹ y *Grammatical Error Correction (GEC)*². Aquí, el término “gramatical” no se utiliza en el sentido descriptivo, sino en el tradicional, ya que se busca corregir errores de todo tipo, incluyendo léxicos, de ortografía o de sintaxis (Bryant, Felice, Andersen, y Briscoe, 2019). Este campo no está definido de manera muy precisa y distintos investigadores han usado el término para referirse a tareas relacionadas entre sí pero no iguales.

De acuerdo con Östling, Gillholm, Kurfali, Mattson, y Wirén (2023), se pueden distinguir dos grupos de correctores: aquellos que se centran en la detección y corrección de errores individuales, priorizando la gramaticalidad del texto, y aquellos que priorizan la naturalidad del texto buscando hacer correcciones a nivel oracional.

Las competencias que presentamos a continuación están dentro del primer grupo. En todas ellas se buscó detectar o corregir errores individuales para mejorar la gramaticalidad de los textos de entrada, principalmente con el propósito de ayudar a hablantes de inglés como segunda lengua. Como se verá más adelante, nuestro proyecto también está dentro del primer grupo, sin embargo, el procedimiento utilizado lo acerca al segundo grupo, ya que, si bien en el texto que retornamos marcamos cada error y corrección de forma individual, el corrector que utilizamos es un LLM, que, como se mostrará, funciona a nivel oracional.

2.1. Competencias anteriores

Así como en otras áreas de la informática y tecnología, en el área de PLN se realizan a veces competencias con un objetivo compartido donde distintos equipos atacan de forma independiente un mismo problema, planteado por los

¹Detección de Errores Gramaticales

²Corrección de Errores Gramaticales

organizadores de la competencia. En particular, respecto de la tarea de detección y corrección de errores en textos, se han realizado recientemente cuatro competencias (CoNLL-2013, CoNLL-2014, BEA-2019 y MultiGED-2023) en las cuales este problema se atacó con diversos enfoques, y a través de las cuales se puede observar una clara evolución en las técnicas para resolverlo.

Para realizar este proyecto nos apoyamos en los resultados y los trabajos de equipos que participaron en las cuatro competencias. Sin embargo, la tarea que atacamos nosotros difiere sustancialmente de las planteadas en las competencias principalmente por el origen del corpus: mientras que en las competencias se buscaba corregir textos generalmente bien escritos pero que presentan algunos errores, nuestro corpus contiene en su mayoría textos con muchos errores. Estos errores además no son los típicamente cometidos por hablantes nativos o avanzados de la lengua sino errores típicos de estudiantes de una segunda lengua de niveles iniciales.

Más allá de estas diferencias, la tarea sigue siendo compartida entre las competencias y nuestro proyecto, por lo que consideramos útil y necesario el análisis de los métodos utilizados y los resultados obtenidos por los equipos participantes.

2.1.1. CoNLL-2013

En la decimoséptima conferencia de *Computational Natural Language Learning*³ (CoNLL-2013) (Ng, Wu, Wu, Hadiwinoto, y Tetreault, 2013) se planteó una competencia para corregir errores gramaticales en textos escritos por personas que aprenden inglés como segundo idioma. Los equipos debían detectar y corregir errores de gramática en ensayos proporcionados, centrados en cinco tipos de errores comunes: artículos/determinantes, preposiciones, número de sustantivos, forma verbal y concordancia sujeto-verbo.

En la competencia CoNLL-2013 los datos principales utilizados fueron los ensayos escritos por estudiantes de pregrado del sudeste asiático en dos temas, estos estudiantes no eran hablantes nativos de inglés. El corpus de entrenamiento fue el NUCLE (NUS Corpus of Learner English), compuesto por 1,414 ensayos con errores corregidos manualmente, mientras que el conjunto de prueba incluyó 50 nuevos ensayos escritos por otros 25 estudiantes también corregidos manualmente. En esta competencia se utilizó la versión 2.3 de NUCLE (versión revisada), que eliminó 17 ensayos duplicados que contenían múltiples anotaciones.

El enfoque más común fue el uso de clasificadores específicos para cada tipo de error. Estos clasificadores se basaban principalmente en aprendizaje automático, heurísticas o enfoques híbridos que combinaban ambos.

El equipo ganador, UIUC (Rozovskaya, Chang, Sammons, y Roth, 2013), utilizó el algoritmo de Averaged Perceptron (AP) para corregir los errores de tipo artículos/determinantes, mientras que para el resto de los tipos de errores utilizaron el clasificador Naive Bayes (NB).

³Aprendizaje Computacional de Lenguaje Natural

Este enfoque les permitió lograr un F_1 de 31.2% con las anotaciones originales propuestas para la tarea. Sin embargo, se les permitió a los equipos proponer correcciones alternativas, es decir, ediciones al gold-standard, y en este caso alcanzaron un F_1 de 42.14%.

En las tablas 2.1 y 2.2 se muestran los resultados de todos los participantes, en el primer caso con las anotaciones originales propuestas para la tarea y en el segundo con anotaciones revisadas a partir de propuestas de correcciones de los participantes.

Rank	Team	R	P	F_1
1	UIUC	23.49	46.45	31.20
2	NTHU	26.35	23.80	25.01
3	HIT	16.56	35.65	22.61
4	NARA	18.62	27.39	22.17
5	UMC	17.53	28.49	21.70

Tabla 2.1: Resultado de la competencia CoNLL-2013 con anotaciones originales

Rank	Team	R	P	F_1
1	UIUC	31.87	62.19	42.14
2	NTHU	34.62	30.57	32.46
3	UMC	23.66	37.12	28.90
4	NARA	24.05	33.92	28.14
5	HIT	20.29	41.75	27.31

Tabla 2.2: Resultado de la competencia CoNLL-2013 con anotaciones revisadas

2.1.2. CoNLL-2014

Un año después de CoNLL-2013, en la CoNLL-2014 (Ng y cols., 2014) la corrección de errores gramaticales fue nuevamente un tema central. Esta vez, se aumentó significativamente la cantidad de tipos de errores a detectar, 28 en lugar de cinco, en ensayos escritos en inglés por estudiantes que tienen el inglés como segunda lengua, con el objetivo final de devolver el artículo corregido.

El corpus de entrenamiento fue el mismo que había sido utilizado en la competencia anterior, mientras que el conjunto de evaluación consistió de 50 nuevos ensayos escritos por otros 25 estudiantes pertenecientes a la misma universidad también corregidos manualmente.

El método más común utilizado por los equipos fue un enfoque híbrido que combinaba modelos de lenguaje basados en n-gramas con traducción automática estadística (SMT) y/o enfoques basados en reglas.

El equipo ganador, CAMB (University of Cambridge) (Felice, Yuan, Andersen, Yannakoudakis, y Kochmar, 2014), justamente, utilizó un sistema híbrido que combinaba los tres métodos mencionados.

Team ID	Precision	Recall	F _{0,5}
CAMB	39.71	30.10	37.33
CUUI	41.78	24.88	36.79
AMU	41.62	21.40	35.01
POST	34.51	21.73	30.88
NTHU	35.08	18.85	29.92
RAC	33.14	14.99	26.68
UMC	31.27	14.46	25.37
PKU	32.21	13.65	25.32
NARA	21.57	29.38	22.78
SJTU	30.11	5.10	15.19
UFC	70.00	1.72	7.84
IPN	11.28	2.85	7.09
IITB	30.77	1.39	5.90

Tabla 2.3: Resultado de la competencia CoNLL-2014

Team ID	Precision	Recall	F _{0,5}
CUUI	52.44	29.89	45.57
CAMB	46.70	34.30	43.55
AMU	45.68	23.78	38.58
POST	41.28	25.59	36.77
UMC	43.17	19.72	34.88
NTHU	38.34	21.12	32.97
PKU	36.64	15.96	29.10
RAC	35.63	16.73	29.06
NARA	23.83	31.95	25.11
SJTU	32.95	5.95	17.28
UFC	72.00	1.90	8.60
IPN	11.66	3.17	7.59
IITB	34.07	1.66	6.94

Tabla 2.4: Resultado de la competencia CoNLL-2014 con respuestas alternativas

La competencia CoNLL-2014 contó con la participación de 13 equipos de todo el mundo, en la tabla 2.3 se muestran los resultados de todos ellos. El mejor sistema en la competencia logró un puntaje F_{0,5} (2.3.3) de 37.33% cuando se considera el *gold standard* original de la competencia. Sin embargo, al igual que en CoNLL-2013, se permitió a los equipos proponer correcciones alternativas, y de esta forma el mejor equipo alcanzó un puntaje de 45.57%.

Aunque los resultados de estas dos competencias pueden parecer pobres, es importante tener en cuenta que tuvieron lugar en una etapa temprana en el desarrollo de los modelos de lenguaje y los sistemas de corrección gramatical automatizados.

Según se reporta en las actas de estas competencias, la tarea resultó muy

desafiante dado que aún se requería una investigación considerable en esta área en aquel momento. Sin embargo, también se reconoce desde ese momento la importancia de la tarea y de las ventajas de desarrollar correctores gramaticales automatizados.

2.1.3. BEA-2019

La tarea compartida de corrección de errores gramaticales en el workshop [BEA](#) (Building Educational Applications) 2019 ([Bryant y cols., 2019](#)) continuó la tradición de las tareas anteriores de [CoNLL](#).

En [BEA-2019](#), se introdujo un nuevo conjunto de datos a estas competencias, el Cambridge English Writing and Progress (W & I) y el LOCNESS Corpus, que representa una variedad más amplia de habilidades en inglés que el corpus utilizado en [CoNLL-2014](#) (NUCLE, compuesto por 1,414 ensayos con errores corregidos manualmente). Mientras que el conjunto de pruebas de esa competencia solo contiene 50 ensayos, el conjunto de pruebas W & I + LOCNESS contiene alrededor de 350 ensayos sobre 50 temas escritos por 334 autores de todo el mundo, incluidos hablantes nativos de inglés. Esto amplía significativamente la diversidad y representatividad del corpus de prueba.

Otra diferencia con las tareas previamente compartidas es la introducción de niveles de recursos para los participantes, que se divide en tres: recursos limitados, ilimitados y bajos. Esto permite controlar el acceso a los datos anotados:

- Limitado: se especifica los conjuntos de datos de aprendizaje anotados que están disponibles para los participantes
- Ilimitado: se permite el uso de cualquier conjunto de datos disponible
- Bajo: se limita significativamente la cantidad de datos anotados disponibles

Con esto, se busca fomentar el estudio de la corrección gramatical en idiomas que carecen de un corpus de entrenamiento anotado.

El método más utilizado en el workshop de [BEA-2019](#) fue el enfoque de *Neural Machine Translation* (NMT)⁴ utilizando [transformers](#) ([Vaswani y cols., 2017](#)), implementado por aproximadamente dos tercios de los equipos. Esto contrasta con los enfoques previos de corrección gramatical ([GEC](#)), que utilizaban reglas, SMT y modelos de lenguaje basados en n-gramas.

El grupo más destacado fue UEDIN-MS ([Grundkiewicz, Junczys-Dowmunt, y Heafield, 2019](#)). Su enfoque fue generar oraciones con errores artificiales. Para esto tomaron 100 millones de oraciones del corpus WMT News Crawl ([Bojar y cols., 2018](#)) e introdujeron errores basándose en la distribución de errores del corpus W&I+LOCNESS. Con estos datos se entrenó un conjunto de sistemas de NMT.

Como se puede observar en la tabla [2.5](#), la estrategia de NMT resultó efectiva, y los equipos en general obtuvieron puntajes significativamente más altos en [BEA-2019](#) que en la tarea compartida anterior de [CoNLL-2014](#).

⁴Traducción Automática Neuronal

Rank	Teams	TP	FP	FN	P	R	F _{0,5}
1	UEDIN-MS	3127	1199	2074	72.28	60.12	69.47
2	Kakao&Brain	2709	894	2510	75.19	51.91	69.00
3	LAIK	2618	960	2671	73.17	49.50	66.78
4	CAMB-CLED	2924	1224	2386	70.49	55.07	66.75
5	Shuyao	2926	1244	2357	70.17	55.39	66.61
6	YDGEC	2815	1205	2487	70.02	53.09	65.83
7	ML@IITB	3678	1920	2340	65.70	61.12	64.73
8	CAMB-CUED	2929	1459	2502	66.75	53.93	63.72
9	AIP-Tohoku	1972	902	2705	68.62	42.16	60.97
10	UFAL	1941	942	2867	67.33	40.37	59.39
11	CVTE-NLP	1739	811	2744	68.20	38.79	59.22
12	BLCU	2554	1646	2432	60.81	51.22	58.62
13	IBM	1819	1044	3047	63.53	37.38	55.74
14	TMU	2720	2325	2546	53.91	51.65	53.45
15	qiuwenbo	1428	854	2968	62.58	32.48	52.80
16	NLG-NTU	1833	1873	2939	49.46	38.41	46.77
17	CAI	2002	2168	2759	48.01	42.05	46.69
18	PKU	1401	1265	2955	52.55	32.16	46.64
19	SolomonLab	1760	2161	2678	44.89	39.66	43.73
20	Buffalo	604	350	3311	63.31	15.43	39.06
21	Ramaiah	829	7656	3516	9.77	19.08	10.83

Tabla 2.5: Resultado de la competencia BEA-2019

2.1.4. MultiGED-2023

MultiGED-2023 (Volodina y cols., 2023) fue una competencia organizada por el grupo de trabajo de SLA Computational⁵, un grupo de trabajo enfocado en el desarrollo de herramientas para promover el aprendizaje de lenguas menos representadas en el área de detección y corrección de errores gramaticales.

Esta competencia se alineó con la tendencia observada en competencias anteriores como las mencionadas en secciones anteriores, donde diferentes equipos abordaron de forma independiente el mismo problema planteado por los organizadores.

A diferencia de las competencias mencionadas en nuestra introducción, y alineado con el objetivo de SLA Computational, MultiGED-2023 buscó contrarrestar la tendencia creciente en la investigación de PLN hacia la dominancia del inglés. Esta competencia buscó equilibrar la dinámica actual en el campo de PLN, proporcionando una plataforma para que investigadores trabajen en idiomas como el checo, ruso, español, ucraniano, rumano, alemán y sueco, entre otros.

En esta competencia solo se trabajó la detección de errores y no la corrección de errores. Además, debido a la dificultad de generar categorizaciones de errores

⁵<https://spraakbanken.gu.se/en/compsla>

válidas para los distintos idiomas en los que se trabajó, la detección es binaria y no en clases. Esto implica que, si bien es la más próxima temporalmente a este proyecto y, por lo tanto, en la que se utilizaron métodos más modernos, la tarea realizada difiere sustancialmente con la que realizamos nosotros.

El método más común utilizado fue el uso de LLMs similares a BERT. Los dos mejores grupos, EliCoDe y DSL-MIM-HUS, utilizaron el modelo de lenguaje XLM-RoBERTa, preentrenado con aproximadamente 100 idiomas. Sin embargo, EliCoDe generó modelos independientes para cada idioma de la competencia mientras que DSL-MIM-HUS generó un solo modelo para todos. En el caso particular del inglés, también fueron estos dos equipos los que obtuvieron mejores puntajes, en la tabla 2.6 se pueden ver los resultados de todos los equipos. Es interesante notar que, si bien la tarea era más sencilla que la planteada en BEA-2019, el mejor resultado fue ligeramente más bajo.

Rank	Teams	P	R	F _{0,5}
1	EliCoDe	73.64	50.34	67.40
2	DSL-MIM-HUS	72.36	37.81	61.18
3	Brainstorm Thinkers	70.21	37.55	59.81
4	VLP-char	20.76	29.53	22.07
5	NTNU-TRH	81.37	1.84	8.45

Tabla 2.6: Resultado de la competencia MultiGED-2023 para idioma inglés

2.1.5. Proyecto de Grado: Construcción de herramientas de soporte para corrección en enseñanza de inglés

Para nuestro trabajo, nos basamos en lo presentado en Brown, Paez, Herrera, Chiruzzo, y Rosá (2023), en este proyecto se utilizó el mismo corpus que disponemos nosotros, descrito en la sección 2.2.2, para generar una herramienta de corrección automática basada en heurísticas. Esta herramienta utilizaba una serie de heurísticas para marcar errores en los textos. También se clasificaba a los textos con un puntaje a través de un algoritmo de aprendizaje automático supervisado, basándose en los errores detectados.

En este proyecto, se consideraron varios tipos de error inicialmente pero para la herramienta generada se consideran ocho: errores de ortografía, omisión de mayúscula al comienzo de oración, omisión de mayúscula en pronombre “I”, omisión de mayúscula en nombres propios, problemas de concordancia sujeto-verbo, forma del verbo, artículo faltante y artículo innecesario.

Tanto en dicho trabajo como en el nuestro, se busca como objetivo macro implementar una herramienta para potenciar la enseñanza de inglés en nuestro país, y ambos usamos el que, de momento, es el único corpus disponible para generar estas herramientas. Sin embargo, los proyectos difieren en cuanto a las herramientas que se utilizan para lograrlo. Como se detalla en Brown y Páez (2021), en ese proyecto se implementó un pipeline para primero pre-procesar el texto y luego corregirlo siguiendo una serie de heurísticas. Además se creó

una herramienta de machine learning tradicional para puntuar los textos según los errores encontrados. Por otro lado, nosotros buscamos aprovechar la potencia de los LLMs para atacar la tarea de detectar y corregir errores en los textos. Además, no buscamos en nuestro caso clasificarlos de ninguna manera, limitándonos a la tarea de corrección y marcado.

En términos generales, el corrector en [Brown y Páez \(2021\)](#) funcionaba como una serie de correctores independientes, cada uno de los cuales era responsable de detectar y marcar un solo tipo de error. Antes de ejecutar estos correctores el texto es pre-procesado para eliminar la mayor cantidad posible de “ruido”: espacios innecesarios, espacios faltantes, formas alternativas pero válidas de escribir algunas palabras. En nuestro caso, el proceso de corrección es diferente ya que usamos una herramienta que no se parece a la de este trabajo, sin embargo, también pre-procesamos el texto de forma similar a modo de estandarizar el input que recibe el corrector.

En el proyecto anterior, se consideró que se logró generalizar bien, sin sobreajustar a los datos de entrenamiento (como se explica más adelante, esto es un riesgo debido a la naturaleza de esos datos), mostrando diferencias menores al ser evaluados en los conjuntos de validación y prueba. Los resultados reportados son más positivos con errores ortográficos, sobre todo en lo que respecta al uso de mayúsculas pero son más bajos en errores sintácticos, especialmente en cuanto al uso de artículos.

2.2. Corpus

En esta sección se describen los corpus disponibles para este proyecto y en particular el corpus principal que utilizamos finalmente.

2.2.1. Corpus disponibles

En el inicio del proyecto analizamos y consideramos los siguientes corpus, que fueron utilizados en las competencias ya presentadas. Finalmente, por lo puesto más adelante, no los utilizamos para este proyecto.

- Cambridge English Write & Improve ⁶: El corpus está compuesto por textos de una página web en línea que ayuda a los estudiantes no nativos de inglés a mejorar su escritura.
- FCE v2.1: El corpus First Certificate in English (FCE) es un subconjunto del Cambridge Learner Corpus (CLC) que contiene 1,244 respuestas escritas a preguntas de examen del FCE. ([Yannakoudakis, Briscoe, y Medlock, 2011](#))
- Lang-8 Corpus of Learner English: Lang-8 es un sitio web de aprendizaje de idiomas en línea que fomenta que los usuarios corrijan la gramática de los demás. ([Mizumoto, Komachi, Nagata, y Matsumoto, 2011](#))

⁶<https://writeandimprove.com/>

- NUCLE: El National University of Singapore Corpus of Learner English (NUCLE) consta de más de 1,400 ejemplos escritos principalmente por estudiantes universitarios asiáticos en la Universidad Nacional de Singapur. (Dahlmeier, Ng, y Wu, 2013)
- LOCNESS: Es un corpus de trabajos escritos en inglés nativo, compuesto por trabajos escritos de estudiantes liceales avanzados británicos, trabajos escritos de estudiantes universitarios británicos y trabajos escritos de estudiantes universitarios estadounidenses con un número total de 324,304 palabras. (Granger, 1998)

2.2.2. Corpus utilizado para el proyecto

Los corpus presentados a continuación contienen textos escritos por estudiantes de inglés de Uruguay, en las condiciones mencionadas previamente, y corregidos de forma manual o automática según se indica. Estos son los datos que utilizamos a lo largo del proyecto.

- Corregidos de forma automática: 54.007 textos corregidos por el pipeline presentado en Brown y Páez (2021), este corpus automático solo fue utilizado para tomar ejemplos nuevos para el [prompt](#) en A.2.6, presentado en la sección 3.7.2.
- Corregidos por graduandos: 53 textos corregidos por los autores de Brown y Páez (2021), este es el corpus con el que trabajamos a lo largo de nuestro proyecto y por lo tanto es nuestro corpus de desarrollo.
- Corregidos por tutores del proyecto: 40 textos corregidos de forma manual, que utilizamos al final para validar la solución obtenida, por lo que es nuestro corpus de test.

Estos corpus contienen anotaciones que se alinean con los objetivos de nuestro estudio, proporcionando un conjunto de datos relevante y específico para nuestros propósitos. Como se observa, la mayor parte del corpus está anotado de forma automática y hay apenas cerca de 100 textos corregidos de forma manual, que contienen alrededor de 600 errores marcados.

Aunque existen otros corpus disponibles, como los proporcionados por las dos competencias de CoNLL y BEA 2019 (los primeros 4 items mencionados en Corpus disponibles), no los utilizamos en nuestro proyecto por varias razones. En particular, estos corpus suelen contener datos de hablantes con un nivel de inglés más alto, y los errores presentes no se asemejan a los que observamos en nuestro corpus. Además, la forma de marcar errores y las etiquetas utilizadas son diferentes a las disponibles en el corpus que utilizamos, lo cuál dificulta su inclusión.

No obstante, una desventaja de utilizar únicamente nuestro corpus es la falta de ejemplos variados, ya que todos los datos provienen de un mismo ejercicio. Esta limitación implica que los errores detectados y, por lo tanto, los resultados obtenidos estarán sobreajustados al tipo de errores presentes en el corpus, por lo

Nombre	Corpus automático	Corpus de desarrollo	Corpus de test
Descripción	Corregidos de forma automática en proyecto anterior	Corregidos a mano por graduandos	Corregidos a mano por tutores
Total textos	54.007	53	40
Puntaje 0	13.746	0	0
Puntaje 1	11.428	10	0
Puntaje 2	17.054	10	9
Puntaje 3	10.281	10	9
Puntaje 4	1.350	10	10
Puntaje 5	135	10	10
Puntaje 6	13	3	2

Tabla 2.7: Textos disponibles en cada corpus por puntaje

cual su desempeño será menor para otros ejercicios o en otros contextos. Como se explica más adelante, por la metodología que utilizamos, este problema se reduce parcialmente pero de todas formas es importante considerar la incorporación de ejemplos más diversos en estudios futuros para mejorar el poder de generalización y la robustez de nuestro modelo de corrección de texto.

Descripción de los datos

El corpus contiene alrededor de 55.000 textos escritos por estudiantes jóvenes de niveles iniciales. Todos responden a una única consigna, que se muestra en la figura 2.1.

La consigna formaba parte de una prueba que incluía además preguntas de múltiple opción. Estas preguntas fueron corregidas en su totalidad por profesores de inglés y a cada prueba se le asignó un puntaje de 0 a 6 basado en las respuestas a esas preguntas y a la respuesta a la consigna escrita. Si bien nosotros no utilizamos los puntajes dados en este proyecto, es importante observar que estos no surgen solo del ejercicio escrito sino de toda la prueba, por lo que su utilidad es limitada. En la tabla 2.7 se muestran los textos disponibles en cada uno de los corpus para cada puntaje, como se puede observar, los datos están muy desbalanceados.

Para este proyecto utilizamos únicamente los textos con puntaje mayor o igual a dos, ya que en general los textos con menor puntaje no estaban en inglés o no contienen suficiente texto para ser útiles en este trabajo.

En la tabla 2.8, listamos todos los tipos de errores existentes en ambos. Como se observa, algunos tipos de errores solo aparecen en el corpus de test. Esto se debe a que los corpus fueron corregidos con distintos criterios y el corpus de test fue corregido de forma más exhaustiva. Como explicamos más adelante,

Tipo de error	Descripción	Corpus de desarrollo	Corpus de test
E_S	Ortografía	72	72
E_C-BOS	Mayúscula al principio de la oración	69	68
E_VF	Forma del verbo	60	38
E_SVA	Concordancia entre sujeto y verbo	45	14
E_MA	Artículo faltante	31	2
E_UA	Artículo innecesario	12	12
E_C-PN	Mayúscula en nombre propio	11	5
E_C-PI	Mayúscula en pronombre "I"	8	0
E_OE	Otros	2	25
E_MV	Verbo faltante	2	2
E_MS	Sujeto faltante	1	18
E_P	Puntuación	0	27
E_WS	Sujeto incorrecto	0	17
E_WV	Verbo incorrecto	0	11
E_MPT	Preposición faltante	0	7
E_NN	Error de número	0	5
E_PR	Error de pronombre	0	4
E_WP	Preposición incorrecta	0	3
E_MC	Conjunción faltante	0	3
E_IWO	Orden incorrecto	0	1
E_T	Error de tipeo	0	1
Total		311	335

Tabla 2.8: Errores en cada corpus por puntaje



Mira la imagen. Esta es Andrea. Escribe en inglés en la casilla todo lo que puedas sobre ella. Presta atención a la foto y a los dibujos: puedes escribir sobre su edad; su aspecto físico; la ropa que está vistiendo en la foto; lo que le gusta y no le gusta; lo que tiene y lo que no tiene; lo que puede y no puede hacer; sus rutinas; etc.

Figura 2.1: Consigna del ejercicio

en este proyecto utilizamos únicamente los errores presentes en el corpus de desarrollo, y además descartamos algunos por tener muy pocos ejemplos. En la tabla marcamos en **negrita** los tipos con los que efectivamente trabajamos.

Dificultades

Uno de los principales retos que enfrentamos en el análisis de los textos fue la alta frecuencia de errores causados por hablantes con un dominio limitado del inglés.

Por ejemplo, observamos casos en los que los estudiantes escriben *right* en lugar de *write* o *ried* en lugar de *read*, basándose en cómo suenan estas palabras en español. Este tipo de errores, causados por la similitud fonética entre ambos idiomas, son comunes en hablantes de español que están aprendiendo inglés. Sin embargo, estas confusiones no suelen ocurrir en hablantes nativos de inglés o hablantes de otro idioma.

Estos errores presentan un desafío adicional para las correcciones automáticas, ya que requieren un enfoque que tenga en cuenta no solo el contexto del idioma de destino, sino también las particularidades de la lengua de origen de los hablantes.

2.3. Herramientas utilizadas

A continuación se presentan las herramientas y conceptos teóricos que utilizamos a lo largo del proyecto.

2.3.1. Aprendizaje automático supervisado

El aprendizaje automático es una rama de la inteligencia artificial que se enfoca en la creación de programas de computadora que mejoran su desempeño automáticamente a partir de la experiencia (Mitchell, 1997).

En el aprendizaje automático supervisado, el modelo aprende a partir de un conjunto de datos etiquetados, es decir, con ejemplos en los que ya se conocen las respuestas correctas. El objetivo es que el algoritmo pueda generalizar a partir de estos ejemplos y hacer predicciones precisas para datos nuevos (Mohri, Rostamizadeh, y Talwalkar, 2012).

En este proyecto se utiliza aprendizaje supervisado para clasificar las correcciones halladas previamente en las distintas categorías, como se presenta en la sección 3.9.

Etapas de aprendizaje supervisado

El aprendizaje supervisado sigue un proceso estructurado que puede dividirse en varias etapas clave. Estas etapas permiten que el modelo aprenda de los datos etiquetados para hacer predicciones precisas. A continuación, se describen las etapas típicas del aprendizaje supervisado:

- **Recopilación de datos:** La primera etapa es obtener un conjunto de datos representativo, que debe incluir ejemplos etiquetados (es decir, datos de entrada junto con su salida o resultado correcto). Estos datos son fundamentales para entrenar el modelo y deben ser de alta calidad y relevantes para el problema a resolver.
- **Preprocesamiento de los datos:** Los datos crudos a menudo contienen ruido, errores o valores faltantes que pueden afectar el rendimiento del modelo. En esta etapa, los datos se limpian y transforman para hacerlos aptos para el entrenamiento.
- **Selección de características:** En esta etapa, se seleccionan las características más importantes que ayudarán al modelo a hacer predicciones precisas, eliminando características irrelevantes o redundantes.
- **Elección del modelo:** Se selecciona un algoritmo de aprendizaje supervisado apropiado para el problema en cuestión. Los algoritmos pueden variar según la naturaleza de los datos y la tarea a realizar.
- **Entrenamiento del modelo:** En esta etapa, el modelo se entrena utilizando el conjunto de datos de entrenamiento.
- **Evaluación del modelo:** Una vez que el modelo ha sido entrenado, se evalúa utilizando un conjunto de datos de validación, que no se utilizó durante el entrenamiento. Se mide el rendimiento del modelo en función de métricas como el **accuracy** (acierto), **precision** (precisión), **recall** (exhaustividad) y puntaje F_1 , o, en nuestro caso, $F_{0,5}$ (2.3.3).

2.3.2. Modelos de lenguaje

Los modelos de lenguaje son modelos que asignan una probabilidad a todas las posibles palabras siguientes (Jurafsky y Martin, 2024). Para calcular esta probabilidad, dependiendo del modelo se pueden considerar distintos datos, por ejemplo cuáles son las palabras previas, *word embeddings* u otro contexto. Se pueden utilizar para procesar o generar texto y se utilizan en diversas tareas, como por ejemplo reconocimiento de voz, traducción automática y recuperación de información.

En particular en este proyecto, utilizamos grandes modelos de lenguaje (LLMs), que son la iteración más reciente de estos modelos. Los modelos de esta clase son pre-entrenados con cantidades enormes de texto, por lo que disponen de muchísima información, lo cual, en teoría, les permitiría hacer predicciones más correctas respecto de la palabra siguiente en un texto.

Los tres modelos que utilizamos a lo largo de este proyecto son ChatGPT⁷ (al momento de escribir está disponible la versión 4o pero nosotros utilizamos la última disponible en su momento, 3,5), LLaMA 2⁸ (utilizamos el modelo 70B a través de una herramienta online e intentamos ejecutar localmente con poco éxito la versión 7B) y Mistral 7B⁹.

Todos estos modelos forman parte del área de *generative AI*¹⁰ (Feuerriegel, Hartmann, Janiesch, y Zschech, 2024). Es decir, son modelos que se utilizan para generar nuevo contenido, en nuestro caso textos. Como dijimos, estos modelos tienen un gran poder predictivo ya que son entrenados utilizando únicamente el contexto previo en cada *token*. Como las probabilidades calculadas durante el entrenamiento no dependen del texto subsiguiente, esos mismos valores se pueden utilizar para predecir la palabra que sigue cuando el texto está incompleto.

La forma en que estos modelos se utilizan para generar texto es condicionándolos a partir de un *prompt*. Es decir, el modelo recibe el *prompt* y predice cómo debe continuar el texto a partir de lo recibido.

Prompt

Un *prompt* es una entrada o conjunto de instrucciones proporcionadas a un LLM para guiar la generación de respuestas. Los *prompts* pueden incluir preguntas, fragmentos de texto o cualquier otra información contextual que ayude al modelo a entender lo que se espera de él.

El uso de *prompts* es esencial para guiar a los LLMs en la generación de respuestas. Un *prompt* bien diseñado puede mejorar significativamente la calidad de las respuestas generadas, ya que proporciona el contexto necesario para que el modelo haga predicciones útiles.

En nuestro trabajo, diseñamos y probamos múltiples *prompts* para optimizar los resultados de corrección de texto. Se ha prestado especial atención a la

⁷<https://openai.com/>

⁸<https://www.llama.com/llama2/>

⁹<https://mistral.ai/news/announcing-mistral-7b/>

¹⁰inteligencia artificial generativa

formulación de [prompts](#) claros y específicos que puedan maximizar la precisión y la relevancia de las correcciones proporcionadas por los [LLMs](#).

Arquitectura de transformers

Diferentes modelos de lenguaje utilizan distintas técnicas para “aprender” a partir de los textos recibidos. Los más simples utilizan modelos de n-gramas, donde las probabilidades de que aparezca una palabra se calculan a partir de las n - 1 palabras anteriores. Los [LLMs](#), sin embargo, utilizan una arquitectura que les permite observar no solo algunas palabras anteriores, sino considerar un contexto mucho más amplio.

Esta arquitectura se denomina *transformers* ([Vaswani y cols., 2017](#)). La forma en que funciona esta arquitectura es que para cada [token](#) del texto, se calcula un *embedding* o representación vectorial, pero esta representación depende del contexto de ese [token](#), es decir, de las representaciones de los [tokens](#) anteriores, y de cómo se relaciona el [token](#) actual con los anteriores. El vector obtenido es utilizado luego nuevamente para calcular la representación de los [tokens](#) posteriores.

En la figura 2.2 se observan los tres componentes principales en la arquitectura de *transformers*. En la fila de más abajo, se encuentra la etapa de codificación de la entrada. En esta etapa se codifica la información semántica del texto en forma de *embedding*, junto con *embeddings* posicionales, que contienen información sobre la posición de la palabra en el texto. Una vez calculado este *embedding*, se pasa a la segunda etapa, donde se encuentran los bloques *transformer*.

Estos bloques contienen cuatro capas. Dos capas son capas de normalización de los datos recibidos, las otras dos son una capa de *multi-head attention* y una de *feedforward*. En la capa *multi-head attention* se recibe el *embedding* del [token](#) actual, así como información perteneciente a los [tokens](#) anteriores, ya sean sus *embeddings* o información procesada por otros bloques. Este es el mecanismo clave por el cual el *transformer* es capaz de procesar información del contexto previo al [token](#) actual (y nunca del contexto posterior). En la capa *feedforward* se analiza la información recibida para hallar relaciones y patrones en los datos recibidos.

Los bloques *transformer* son apilados y aplicados sucesivamente, en cada etapa recibiendo información de las etapas anteriores del mismo [token](#) y también de los [tokens](#) anteriores a través de la capa *multi-head attention*. En cada una de estas etapas, se modifican parámetros que afectan el peso que se le da a los distintos datos. Esto permite que el modelo identifique distintas posibles relaciones entre los [tokens](#).

En el diagrama, se puede observar cómo los *embeddings* de los [tokens](#) anteriores en el texto, así como los resultados de cada bloque *transformer* son utilizados en el procesamiento de los [tokens](#) posteriores. Aunque no se observa en el diagrama, no se utilizan todos los [tokens](#) anteriores al actual sino una ventana de tamaño fijo, pero esta ventana es muy grande en estos modelos, pudiendo llegar incluso a 32768 [tokens](#) ([Jurafsky y Martin, 2024](#)).

Finalmente, en la etapa de modelado de lenguaje, luego de procesar los datos,

se calcula la distribución de probabilidad sobre los **tokens** y se determina cuál es el siguiente **token** a partir de esas probabilidades.

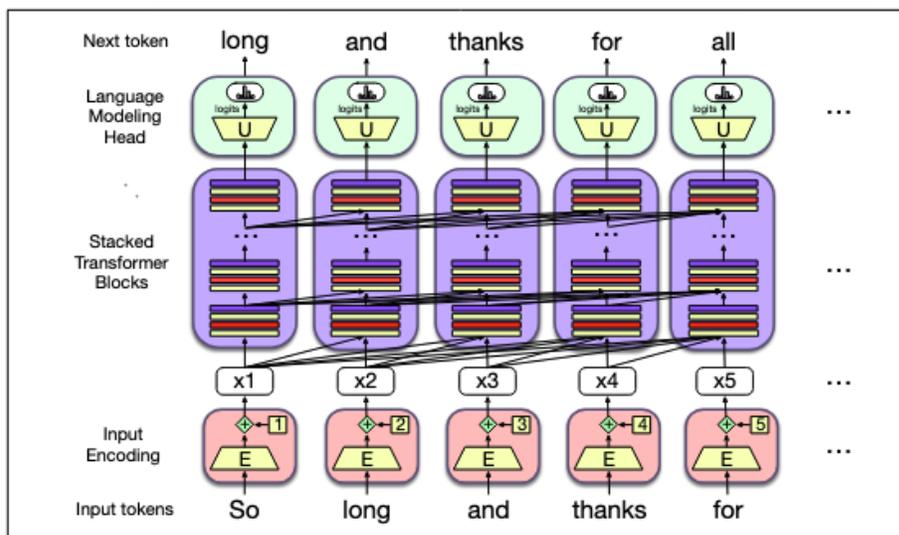


Figura 2.2: Arquitectura de un *transformer* (tomada de [Jurafsky y Martin \(2024\)](#))

Un aspecto fundamental de esta arquitectura es que los cálculos necesarios para cada **token** no dependen de los cálculos hechos para los anteriores, esto permite que esas cuentas sean paralelizadas. La forma de hacer esto es agrupando los distintos grupos de vectores (*embeddings*, vectores de pesos, vectores de parámetros) en matrices y utilizando mecanismos de multiplicaciones de matrices. Estas dos optimizaciones hacen que calcular las probabilidades de cada **token** sea mucho más rápido que con otros modelos y es parte de lo que permite que el modelo observe un contexto tan grande (en el orden de miles de **tokens**).

ChatGPT

ChatGPT es un modelo de lenguaje grande desarrollado por OpenAI. Este modelo se basa en el modelo de lenguaje GPT que fue entrenado con una amplia variedad de datos textuales. Al modelo GPT se le agregaron etapas posteriores de entrenamiento con corpus de consultas y respuestas, de esta forma se logra que genere respuestas coherentes y contextualmente relevantes. ChatGPT ganó popularidad rápidamente desde su lanzamiento en 2022 debido a su capacidad para mantener conversaciones naturales y proporcionar información útil en diversos contextos.

Una de las características distintivas de ChatGPT es que está disponible de forma gratuita en línea a través de la plataforma de OpenAI, lo que permite a los usuarios acceder a sus capacidades sin costo. Sin embargo, a pesar de su

disponibilidad en línea, ChatGPT no puede ser ejecutado localmente.

En este proyecto utilizamos ChatGPT 3.5, la versión gratuita más reciente al momento de usarlo, aunque ya se han liberado versiones más modernas.

LLaMA

LLaMA es un modelo desarrollado por Meta (anteriormente conocido como Facebook). A diferencia de ChatGPT, LLaMA puede ser utilizado de forma gratuita localmente, lo que lo hace accesible para desarrolladores.

Recientemente, Meta ha lanzado LLaMA 3, una versión mejorada que promete avances significativos en capacidad y eficiencia. Sin embargo, nosotros utilizamos LLaMa 2 (Touvron y cols., 2023), ya que era el modelo disponible cuando estábamos en esa etapa del proyecto.

LLaMA 2 está disponible en tres “versiones”: 7B, 13B y 70B, que corresponden con la cantidad de parámetros (en miles de millones) utilizados para su entrenamiento. En este proyecto utilizamos la versión 70B en la sección 3.3 y la versión 7B en la sección 3.7.

Mistral

Mistral (Jiang y cols., 2023) es otro modelo de lenguaje que ha ganado tracción por sus capacidades y flexibilidad. Similar a LLaMA, Mistral permite su uso local sin costo, ofreciendo una alternativa poderosa para aquellos que prefieren ejecutar modelos de lenguaje en sus propios sistemas. Mistral se hizo famoso rápidamente al demostrar que su modelo Mistral 7B supera a LLaMA 7B en todas las métricas, ofreciendo un rendimiento superior en diversas tareas de procesamiento del lenguaje.

Este modelo había sido presentado recientemente durante nuestra investigación y afirmaba tener resultados comparables a LLaMA 13B, siendo ejecutable con recursos comparables a los de LLaMA 7B. Este fue el modelo que finalmente utilizamos en la solución generada.

Al igual que con los otros dos modelos probados, ya hay disponibles nuevas versiones de Mistral que deberían mejorar el desempeño del sistema.

Ambos modelos, LLaMA y Mistral, representan una tendencia hacia la democratización de los LLMs, proporcionando herramientas avanzadas de procesamiento del lenguaje que pueden ser utilizadas y adaptadas por una comunidad más amplia de usuarios y desarrolladores.

2.3.3. Métricas

Como mencionamos previamente, para evaluar el desempeño de los modelos entrenados en aprendizaje supervisado, se utilizan diversas métricas estándar que permiten cuantificar su capacidad para realizar predicciones correctas.

Conteos de tokens marcados

- True Positives (TP): Tokens marcadas como errores que fueron correctamente identificados y corregidos por un modelo.
- False Positives (FP): Tokens correctas que fueron marcadas incorrectamente como errores.
- False Negatives (FN): Tokens marcados como errores en el standard y que no fueron identificados o corregidos por el modelo.
- True Negatives (TN): Tokens correctas que el modelo no marcó incorrectamente.

Precision

Esta métrica indica la proporción de instancias clasificadas como positivas que son realmente positivas. Se calcula como la cantidad de verdaderos positivos dividida por la suma de verdaderos positivos y falsos positivos.

La fórmula de Precisión se define como:

$$\text{Precisión} = \frac{TP}{TP + FP}$$

Donde:

- TP es el número de verdaderos positivos.
- FP es el número de falsos positivos.

Recall

Esta métrica indica la proporción de instancias positivas que han sido correctamente clasificadas como positivas. Se calcula como la cantidad de verdaderos positivos dividida por la suma de verdaderos positivos y falsos negativos.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Donde:

- TP es el número de verdaderos positivos.
- FN es el número de falsos negativos.

F_1 y $F_{0,5}$

$F_{0,5}$ enfatiza la precisión el doble que la [recall](#), mientras que F_1 pondera la precisión y la [recall](#) por igual. Cuando un corrector gramatical se utiliza en la práctica, es crucial que las correcciones propuestas sean altamente precisas para ser aceptadas por los usuarios. No proponer una corrección no es tan perjudicial como proponer una corrección errónea (Ng y cols., 2014). En las competencias

presentadas posteriores a CoNLL-2014 inclusive, así como en nuestro proyecto, se utiliza $F_{0,5}$.

Distancia de Levenshtein

La distancia de Levenshtein es una métrica utilizada para medir la diferencia entre dos cadenas de caracteres. También conocida como “distancia de edición”, representa el número mínimo de operaciones necesarias para transformar una cadena en otra. Estas operaciones incluyen:

- Inserción: Agregar un carácter.
- Eliminación: Quitar un carácter.
- Sustitución: Reemplazar un carácter por otro.

El algoritmo se puede ver en el anexo A.1.

Se puede observar que en la figura 2.3, para transformar la palabra relevant en elephant, se han realizado las siguientes operaciones con un costo unitario asignado para cada inserción, eliminación y sustitución:

		E	L	E	P	H	A	N	T
	0	1	2	3	4	5	6	7	8
R	1	1	2	3	4	5	6	7	8
E	2	1	2	2	3	4	5	6	7
L	3	2	1	2	3	4	5	6	7
E	4	3	2	1	2	3	4	5	6
V	5	4	3	2	2	3	4	5	6
A	6	5	4	3	3	3	3	4	5
N	7	6	5	4	4	4	4	3	4
T	8	7	6	5	5	5	5	4	3

Figura 2.3: Un ejemplo aplicando el algoritmo

- Eliminación: Se elimina la letra ‘r’ de la palabra original.

- Inserción: Se inserta la letra 'p' en la posición correspondiente.
- Sustitución: La letra 'v' se sustituye por 'h', lo que implica un costo unitario.

Este ejemplo utiliza la distancia de edición para una palabra, tomando los caracteres como unidades de trabajo. En nuestra solución en algunos casos lo usamos así y en otros buscamos la distancia de edición para un texto, tomando los **tokens** como unidades de trabajo. Además, en algunos casos, presentados oportunamente, modificamos los costos de ciertas operaciones para obtener resultados más adecuados a lo que buscábamos.

Word Error Rate(**WER**)

WER es una medida comúnmente utilizada en la evaluación de sistemas de reconocimiento del habla o de traducción automática. En el contexto de alineación de texto, el **WER** se utiliza para comparar dos textos y medir la discrepancia entre ellos en términos de palabras.

El cálculo del **WER** implica contar el número de inserciones, eliminaciones y sustituciones necesarias para convertir un texto de referencia en un texto de salida generado por un sistema de reconocimiento automático. Estos errores se dividen por el número total de palabras en el texto de referencia para obtener un porcentaje de error.

La Word Error Rate (**WER**) se calcula con la fórmula:

$$WER = \frac{S + D + I}{N}$$

Donde:

- *S* es el número de sustituciones.
- *D* es el número de eliminaciones.
- *I* es el número de inserciones.
- *N* es el número total de palabras en la referencia.

Un **WER** más bajo indica una mayor similitud entre los textos, lo que sugiere una mejor calidad de alineación o transcripción.

Capítulo 3

Descripción de la solución

En esta sección se introduce el problema central abordado en esta investigación, que consiste en la detección y corrección de errores comunes en la escritura en inglés de estudiantes de nivel inicial. Además, se presenta la propuesta inicial de solución, la cual busca mejorar la precisión y eficacia en la corrección automática de estos errores.

3.1. Descripción de la problemática

Aunque el objetivo general de este problema es compartido con las competencias presentadas anteriormente, se diferencia de estas en algunos aspectos importantes.

Como se explicó, los corpus utilizados en dichas competencias suelen ser textos escritos por hablantes nativos o con un nivel avanzado del idioma inglés. Debido a esto, los textos suelen estar generalmente bien escritos, con algunos errores. Por otro lado, este proyecto surge a partir de la necesidad de corregir textos escritos por estudiantes jóvenes (niños y adolescentes) con niveles iniciales en la lengua. Esto hace que los textos disponibles tengan muchos errores generalmente y además que estos errores sean diferentes a los hallados en otros corpus. Un ejemplo muy común de errores típicos de nuestros textos es la falta de puntuación o la falta de letras mayúsculas. También aparecen errores típicos de la interferencia con el español como la ausencia de sujeto en la oración, e incluso palabras o textos completos en español.

Otra diferencia clave es el tamaño del corpus disponible. Si bien en trabajos anteriores también es importante la dificultad para crear corpus anotados grandes porque exigen mucho tiempo y esfuerzo, en nuestro caso este problema se exagera porque se dispone de textos que responden a una única consigna, y que, además, por las características de la consigna y los estudiantes, son respuestas similares entre sí y cortas. Esto reduce la cantidad de errores diferentes disponibles. A esto se le suma que la parte del corpus que está anotada es menor aún y el corpus de desarrollo no comparte todos los criterios con el corpus de

test.

Con todas estas consideraciones, la problemática a abordar es la corrección e identificación de errores en textos, en el contexto de estudiantes de inglés de niveles iniciales cuya lengua madre es español. Si bien, como se verá, por el enfoque que tomamos, este contexto no resultó tan importante a la hora de corregir, sí lo fue en la etapa de identificación de errores.

3.2. Propuesta de solución

Desde el comienzo se planteó investigar la posibilidad de usar [LLMs](#) para atacar este problema. Investigamos la posibilidad de utilizar solo [LLMs](#) en todo el proceso pero dados los resultados que obtuvimos (presentados más adelante), finalmente tomamos otro camino: proponemos una herramienta que, en primer lugar, corrige los textos utilizando [LLMs](#), y luego a partir de los textos originales, los textos corregidos y el corpus anotado disponible, encuentra, identifica y marca las correcciones hechas por el modelo.

Esta propuesta se diferencia de otros enfoques dado que no procesamos el texto original para encontrar errores sino que los errores surgen de comparar ese texto con la versión corregida que recibimos del [LLM](#).

Para llevar a cabo el proyecto, analizamos distintos [LLMs](#), cómo ejecutarlos y cuál es la mejor forma de obtener correcciones útiles. Además experimentamos con diferentes configuraciones para generar un clasificador de errores como se describe en las secciones siguientes.

3.3. Primer enfoque con [LLMs](#)

Con el fin de explorar el uso de [LLMs](#) en nuestro proyecto, se comenzó evaluando su capacidad para realizar correcciones gramaticales de manera automática. Se utilizó un conjunto de textos de prueba con el objetivo de observar cómo herramientas como ChatGPT y LLaMA identificaban y corregían errores.

3.3.1. Examinar la mejor versión de [prompt](#) en ChatGPT

Como primer enfoque, se intentó optimizar el rendimiento de los [LLMs](#) mediante la búsqueda de la mejor versión de [prompt](#). Esta etapa inicial fue crucial para ajustar de forma más precisa. Se llevó a cabo dicha prueba en ChatGPT, aprovechando la facilidad que ofrece su plataforma para este tipo de evaluaciones. Sin embargo, no se realizó la misma prueba en LLaMA, pero se reutilizó el mismo [prompt](#) para dicho modelo en fases posteriores.

Se utilizó una versión inicial del [prompt](#) (A.2.1), proporcionando toda la información posible. En esta prueba se incluyeron los siguientes elementos:

- Una descripción de los distintos tipos de errores, indicando cuándo ocurren y un ejemplo ilustrativo.
- Un ejemplo de corrección posible.

- La frase original que se desea corregir.

Dado que la versión inicial no produjo el resultado esperado, se probó una segunda versión (A.2.2) con un enfoque completamente diferente, eliminando la descripción de los distintos tipos de errores como parte de la entrada. En esta versión, se busca que ChatGPT identifique las diferencias entre el texto original y el corregido, marcando los errores con una etiqueta <error> en la frase original en la salida. El objetivo es destacar los errores específicos en el texto, facilitando así su identificación y corrección.

Sin embargo, la segunda versión arrojó resultados aún peores que la primera. Por ello, la tercera versión del [prompt](#) (A.2.3) se basó nuevamente en la versión inicial, pero con una reducción en la descripción de los tipos de errores, limitándose a proporcionar el tipo de error junto con su respectiva etiqueta, sin incluir información sobre cuándo ocurren. No obstante, se mantuvo la inclusión de un ejemplo de corrección. Este cambio se realizó porque consideramos que el exceso de explicación sobre los distintos tipos de errores y los momentos en que ocurren podría generar confusión en el modelo de lenguaje.

Se eliminó el ejemplo de corrección en la cuarta versión (A.2.4), ya que no se observaron cambios significativos en los resultados obtenidos.

En la versión final (A.2.5), se eliminaron los ejemplos de los distintos tipos de errores, dado que no generaron cambios significativos en los resultados. Esta versión se centró en simplificar al máximo el [prompt](#), manteniendo la misma calidad en la salida.

Con la versión final del [prompt](#), se realizaron pruebas en ambos [LLMs](#) utilizando el mismo conjunto de textos de prueba. Los resultados se detallan en la sección 4.1. Como se puede observar, no fueron satisfactorios, por esto decidimos cambiar el enfoque y utilizar los [LLMs](#) únicamente para corregir, ya que en este aspecto los resultados sí son muy positivos. La parte de etiquetado la resolvimos mediante otras técnicas descritas a continuación.

3.4. Solución final desarrollada

El resultado final es un programa que recibe:

1. Pares de textos [original, corregido], con las correcciones marcadas como se describe más adelante, que utilizamos para entrenar un componente del sistema.
2. Textos originales posiblemente con errores, sin procesar, que luego la solución procesa y corrige.

Para cada texto original en el punto 2 de la lista anterior el programa retorna una versión corregida y con los errores marcados con las etiquetas disponibles.

Como se observa en la figura 3.1, los textos a corregir son preprocesados inicialmente, y luego son procesados por tres módulos independientes:

- Un corrector, es decir un módulo que devuelve una versión del texto de entrada donde ya se realizaron todas las correcciones posibles de acuerdo al modelo, en este caso un [LLMs](#).
- Un marcador. Este módulo compara pares de textos, el original y el corregido, y coloca etiquetas en el segundo de ellos, marcando dónde se han hecho correcciones. Este módulo consta de dos partes, el alineador y el extractor de pares.
- Un clasificador. Finalmente, esta pieza recibe los errores identificados por el módulo anterior y los clasifica en categorías que representan los diferentes tipos de error.

Luego de este proceso, el módulo etiquetador toma los resultados de las partes anteriores y genera las oraciones corregidas y etiquetadas que el sistema devuelve.

3.5. Textos etiquetados

Tanto en los textos corregidos recibidos en la entrada como los devueltos por el programa se utiliza un conjunto de etiquetas para marcar los errores. Este conjunto en principio es estático y no modificable. Si aparecieran etiquetas distintas en los textos de entrada el proceso no funcionaría. Sin embargo, dadas las características del programa, sería posible en un trabajo futuro permitir que el programa tomara como input el conjunto de etiquetas disponibles para aumentar su utilidad.

Las etiquetas que utilizamos son las mismas que están presentes en el corpus anotado y que habían sido utilizadas en el proyecto previo, y corresponden con las presentadas en la sección [2.2.2](#). En la tabla [3.1](#), se detallan algunos ejemplos de los errores más comunes, junto con las etiquetas correspondientes para cada tipo de error identificado en los textos.

Una característica importante a tener en cuenta en los pares de textos tomados como entrada es que todas las diferencias entre original y corregido deben estar entre etiquetas. De lo contrario, es imposible alinear los textos para encontrar los errores existentes. No obstante, esto no se cumple en los textos retornados por el programa ya que en algunos casos, por ejemplo en cambios de puntuación, omitimos etiquetas que de otra forma hubieran hecho mucho ruido.

3.6. Preprocesamiento de textos

Tanto los textos que utilizamos para entrenar nuestro algoritmo como los textos que buscamos corregir provienen del corpus descrito en la sección [2.2.2](#). Sin embargo, no utilizamos todos los textos disponibles ni los utilizamos tal cual los obtuvimos. A continuación describimos el proceso de selección y procesamiento de los textos.

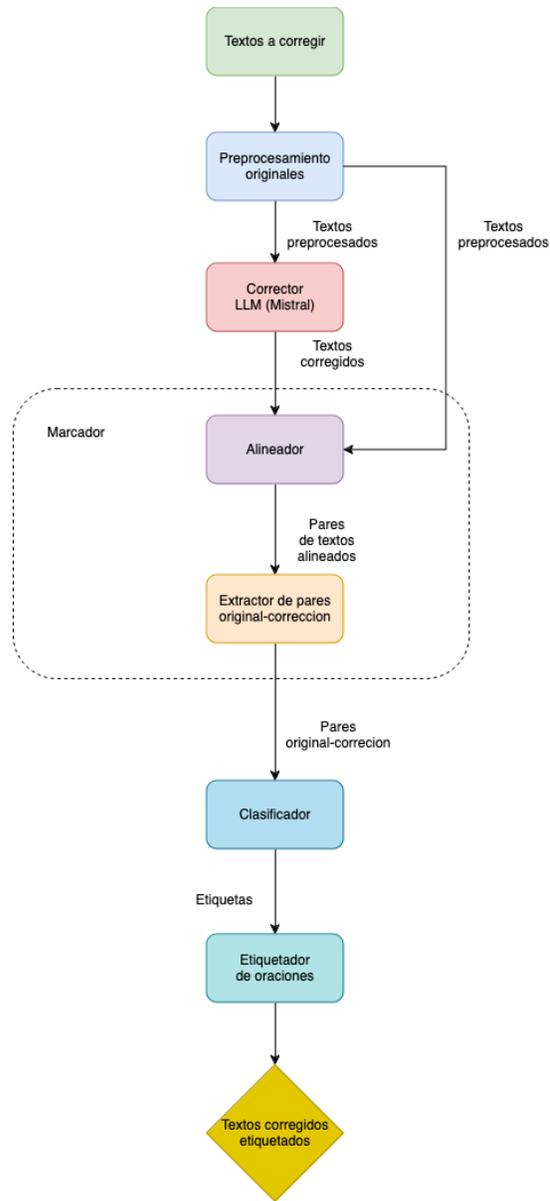


Figura 3.1: Diagrama de la solución

3.6.1. Selección y edición de textos

En primer lugar, hubo ciertos textos que no utilizamos en absoluto. Como se explicó antes, los textos provienen de pruebas que ya fueron clasificadas. En ninguno de los dos corpus utilizados existen pruebas con puntaje 0, y tomamos

Error Tag	Ejemplo
Ortografía	She has got medium <E_S>legth</E_S>, blonde hair.
Mayúscula al comienzo de la oración	<E_C-BOS>she</E_C-BOS>is Andrea.
Mayúscula en pronombre "I"	She is a 14 years old, <E_C-PI>i</E_C-PI>have a dog.
Mayúscula en nombres propios	This is <E_C-PN>andrea</E_C-PN>.
Forma del verbo	Andrea likes to ride a bike and <E_VF>eat</E_VF>.
Concordancia sujeto-verbo	She <E_SVA>have</E_SVA>14 years old.
Artículo faltante	She doesn't like eating <E_MA>apple</E_MA>.
Artículo innecesario	She likes reading a <E_UA>books</E_UA>.

Tabla 3.1: Errores disponibles

la decisión de descartar los textos de pruebas con puntaje igual a uno. Esto es porque esos textos muchas veces no tienen oraciones sino solo palabras sueltas (por ejemplo, "book pezza apple") o frases que no son oraciones ("14 years old", "like fishing"), por lo que son difíciles de utilizar en nuestro proceso.

Por otro lado, algunos textos puntuales fueron editados manualmente por nosotros para evitar errores al momento de alinearlos. En algunos casos notamos que no se cumplía la condición de que las diferencias entre original y corregido estuvieran etiquetados, en otros había errores en las etiquetas (por ejemplo, faltaba una "/" en la etiqueta de cierre).

Finalmente, sacamos algunas etiquetas usadas en el corpus. Como se observa en la tabla 2.8, las etiquetas E_OE (otros errores), E_MS (sujeto faltante) y E_MV (verbo faltante) se usaban una o dos veces en todo el corpus y todas se encontraban en solo dos de los textos del corpus. Las descartamos dado que era contraproducente entrenar un clasificador con tan pocos ejemplos. También eliminamos etiquetas cuando encontramos errores anidados en el corpus original, por ejemplo:

```
shi -> <E_BOS><E_S>She<E_S><E_BOS>
```

Nuestro algoritmo de alineado no soporta este uso por lo que, en esos casos, mantuvimos la etiqueta que consideramos más apropiada.

3.6.2. Preprocesamiento del texto

Más allá de estas ediciones manuales hechas previo a utilizar los textos, también preprocesamos los textos recibidos de forma automática para poder

utilizarlos en los siguientes pasos.

Nuestro preprocesamiento busca estandarizar los textos, los cuales muchas veces tienen errores causados por el uso incorrecto de algunos caracteres, pero que no afectan la gramaticalidad del texto. Dentro de este caso se encuentran:

- Agregado de espacios a continuación de marcas de puntuación
- Sustitución de espacios múltiples por un único espacio
- Eliminación de espacios al comienzo y final del texto
- Sustitución de caracteres similares al apóstrofo (` y `) por el correcto (')

Por último, también hicimos algunos cambios para poder facilitar las tareas siguientes tanto en los textos originales como en los corregidos:

- Agregado de espacios antes y después de las etiquetas de errores
- Sustitución de saltos de línea por espacios

3.7. Corrección de textos utilizando LLMs

Como se muestra en el capítulo de experimentación (4.1), los resultados de utilizar LLMs para corregir y etiquetar los textos no son satisfactorios. Si bien los modelos que probamos obtienen muy buenos puntajes cuando se les pide corregir sin etiquetas, estos puntajes caen drásticamente cuando se les pide que los etiqueten. En algunos casos porque las etiquetas no están en los lugares correctos, en otros porque no son las etiquetas adecuadas. Debido a esto, decidimos utilizar los LLMs únicamente para la tarea de corrección.

3.7.1. Elección y ejecución de LLMs

Como ya se explicó en la sección 3.3, habíamos hecho experimentos previamente con ChatGPT y LLaMA, en ambos casos utilizando versiones disponibles en la web de ambos modelos, accediendo a ellos de forma manual. Para esta parte del trabajo, sin embargo, necesitábamos un modelo que pudiéramos ejecutar libremente, sin depender de otros servicios.

ChatGPT queda automáticamente descartado dado que no es de uso libre y no es posible ejecutar el modelo de forma local. Por otro lado, LLaMA sí es de uso libre pero de todas formas ejecutarlo no es tan sencillo. Los LLMs requieren una gran capacidad de procesamiento y en particular de GPUs¹, lo cual dificulta su ejecución local.

Para ejecutarlo, obtuvimos acceso al Centro Nacional de Supercomputación (ClusterUY)(Nesmachnow y Iturriaga, 2019), el cual dispone de computadoras potentes a las que se puede acceder como servicio. Sin embargo, aun así encontramos dificultades para ejecutarlo. Al momento de escribir esto ya se ha agregado más capacidad y es esperable que aumente más en el futuro, pero cuando

¹unidades de procesamiento gráfico

estábamos llevando a cabo esta parte del proyecto solo había tres computadoras con GPUs adecuadas para este trabajo y solo logramos ejecutar LLaMA 7B en la más grande de ellas. Naturalmente esto limitaba mucho nuestra capacidad de trabajo ya que podía suceder que esta computadora estuviera siendo utilizada por otro usuario.

Para enfrentar este problema buscamos otros modelos que pudiéramos ejecutar con menor cantidad de recursos. El elegido fue Mistral 7B (Jiang y cols., 2023), ya que afirma tener un desempeño superior incluso a LLaMA 13B y logramos ejecutarlo en las tres computadoras de ClusterUY con GPUs adecuadas, por lo que proseguimos el proyecto con este modelo.

Más adelante, pasamos el proyecto a Google Colab² donde también se puede ejecutar este modelo. Sin embargo, es importante notar que el *tier* gratuito no garantiza la disponibilidad del servicio, sino que depende de la demanda en cada momento. Esto no fue un impedimento grave para nosotros, pero consideramos que, en la medida que se utilicen más este tipo de modelos, será necesario atacar el problema de la disponibilidad de recursos para ejecutarlos.

3.7.2. Corrección de textos

Como explicamos, para la corrección de textos en esta etapa utilizamos el modelo Mistral 7B. Este modelo tiene una versión para chat (similar a ChatGPT), pero nosotros utilizamos el modelo normal ya que no necesitábamos esas features. Para entrenarlo, utilizamos few-shot learning³ ya que buscamos que el modelo tuviera disponibles ejemplos de los textos escritos por estudiantes así como distintas clases de correcciones posibles.

Para esto, tomamos tres textos que no habían sido corregidos previamente y los corregimos manualmente, sin etiquetarlos. Los originales y nuestras correcciones están disponibles en A.3. El `prompt` generado (A.2.6) contiene los textos con sus correcciones, en cada caso indicando si es original o corrección y también etiquetas de comienzo y cierre de los textos, y luego, con el mismo formato, el texto que se quiere corregir y la indicación de que a continuación se espera el texto corregido. El modelo es invocado una vez para cada texto a corregir siempre utilizando el mismo `prompt`, por lo que no tiene memoria de las correcciones anteriores.

El resultado final de ejecutar esta parte del proyecto es un conjunto de textos corregidos por el modelo utilizado.

3.8. Marcado de diferencias utilizando WER

Esta etapa se divide en dos tareas, en la primera se alinean los textos utilizando un algoritmo WER basado en la distancia de Levenshtein, y en la segunda se insertan tags *dummy* para marcar las diferencias halladas.

²<https://colab.google/>

³aprendizaje con pocos ejemplos

3.8.1. Alineado de textos

Una vez obtenida la corrección hecha por Mistral, el proceso debe identificar las correcciones hechas por el modelo y luego etiquetarlas con el conjunto de etiquetas. La técnica que utilizamos se presenta en [Swanson y Yamangil \(2012\)](#), consiste en calcular la distancia de Levenshtein para alinear los textos y luego utilizar un clasificador entrenado para etiquetar el tipo de error en las correcciones identificadas.

En primer lugar, para identificar dónde se encontraban las correcciones, alineamos los textos. Utilizamos un algoritmo basado en buscar la distancia de Levenshtein entre las dos oraciones, donde las unidades son sus `tokens`. Es decir, considerando las operaciones de inserción, borrado y sustitución de `tokens`, se busca cuál es el conjunto de operaciones de menor costo total, donde el costo total es la suma de los costos de cada operación.

El algoritmo que utilizamos es una versión apenas modificada del habitual. En lugar de considerar todas las operaciones con costo igual a uno, para obtener el costo de la sustitución aplicamos nuevamente este algoritmo a nivel de los `tokens` a comparar. De esta forma podemos distinguir casos donde dos sustituciones posibles tienen el mismo costo, eligiendo la que es más intuitiva, por ejemplo en el caso de una sustitución de *fishi*ng por *to fish*, hay dos alineaciones equivalentes cuando los costos son todos uno:

fishing		fishing
to fish		to fish
S I		I S

En cambio, podemos calcular el costo de sustitución aplicando el algoritmo [WER](#) a nivel de los caracteres. En el primer caso, el valor [WER](#) para aplicar la sustitución entre *fishi*ng y *to* es 1.0, ya que se ejecutan dos sustituciones y cinco eliminaciones. En el otro caso, la sustitución de *fishi*ng por *fish* consta únicamente de tres eliminaciones, por lo que el costo es de 0.43. En total, la primera opción tiene un costo de 2.0 (1.0 por la sustitución y 1.0 por la inserción) y la segunda opción tiene un costo de 1.43 (0.43 por la sustitución y 1.0 por la inserción), por lo que elegimos la segunda por tener menor costo.

El resultado de aplicar este algoritmo es una lista de operaciones a través de las cuales se puede llegar del original al corregido, es decir que las operaciones son en un sentido las “correcciones” que se le hacen a la oración.

Por ejemplo, a continuación mostramos un texto escrito por un estudiante, la corrección dada por el [LLM](#) y cómo se alinean los dos textos utilizando el algoritmo. El resultado son las operaciones mínimas necesarias para pasar del original a la corrección, donde no se muestran las operaciones “E” que representan que no se debe hacer ningún cambio.

original:	she is andrea she have 14 years old she have one dog
corrección:	She is Andrea . She is 14 years old . She has one dog .
operaciones:	S S I S S I S S I

3.8.2. Insertado de tags

Hay, en realidad, otros detalles que tomar en cuenta, ya que una corrección real puede involucrar varias palabras y no ser, por ejemplo, una sustitución o una inserción simple. En este trabajo tomamos generalmente la heurística de que si se hallaba cualquier cantidad de operaciones seguidas entonces se trataba de una única corrección y quedan por lo tanto, agrupados. No se agrupan operaciones que estén separadas por al menos un `token` sin cambios.

Esta heurística fue tomada de [Swanson y Yamangil \(2012\)](#), en donde se observa que unir operaciones separadas por una o más palabras o no unir operaciones es menos beneficioso que unir solo cuando los cambios son contiguos. Sin embargo, consideramos que en muchos casos esto no es adecuado para nuestra metodología y nuestro corpus. Observamos que en algunos casos (por ejemplo el de `fishing -> to fish` presentado arriba), esta heurística es adecuada, pero en los textos del corpus muchas veces ocurre que se dan muchos errores independientes seguidos. Por ejemplo:

```
she go to fish ->
<E_BOS>She</E_BOS> <E_SVA>goes</E_SVA> <E_VF>fishing</E_VF>
```

Con esta heurística, ese cambio se considera un único error en nuestro proceso, lo cual reduce mucho el puntaje que recibe. Creemos que la heurística podría ser más satisfactoria con textos en donde los errores estén más dispersos.

Una posible mejora sería lo presentado por [Xue y Hwa \(2014\)](#), es decir un clasificador especializado en identificar si corresponde unir operaciones o no. Creemos que dado el tamaño pequeño de nuestro corpus, ese clasificador no sería muy beneficioso y no lo implementamos. Consideramos que, a futuro, una mejora en este aspecto sería beneficiosa para el sistema.

Independientemente de cómo se determine qué operaciones se deben unir o no, el procedimiento que seguimos para marcar errores es insertar una etiqueta *dummy* en donde comienzan las operaciones de edición y su correspondiente etiqueta de cierre donde se considera que terminan. Esto resulta en un texto similar a los textos etiquetados del corpus con la excepción de que todas las etiquetas son iguales.

Sin perjuicio de la heurística general presentada antes, tuvimos en consideración algunas excepciones a la aplicación de esta regla, lo que nos permitió manejar mejor algunos casos particulares.

En primer lugar, consideramos que la puntuación siempre delimita errores. Es decir que, sin importar si la puntuación fue agregado por la corrección, no lo agrupamos ni con los errores precedentes, ni con los siguientes, aislando así tanto a la marca de puntuación como a los errores que la puedan rodear.

También tomamos consideraciones para los casos de contracciones en inglés, es decir *don't*, *doesn't*, *I'm*, etc. Tomamos en cuenta dos casos.

Primero, que los [LLMs](#) a veces cambian el texto original con contracciones eliminándolas y a veces agregan contracciones en textos donde no las había. Es decir, en el texto original puede haber un *do not* que el modelo cambia por *don't* o incluso en algunos casos pasa lo contrario. Estos cambios pueden mejorar la

naturalidad de un texto, pero no suelen provenir de errores gramaticales, por lo que evitamos marcar errores cuando el único cambio es un pasaje de usar contracciones a no usarlas, o viceversa.

En segundo lugar, encontramos dificultades en los casos que sí hay errores en palabras con contracciones, por ejemplo:

- `don't -> <E_SVA>doesn't</E_SVA>`
- `doesnt -> <E_S>doesn't</E_S>`

En estos casos, nuestro tokenizador separa (correctamente) en [tokens](#):

- `["do", "n't"] -> ["does", "n't"]`
- `["does", "nt"] -> ["does", "n't"]`

Por lo que, si analizamos los cambios a nivel de [token](#), encontramos las siguientes correcciones:

- `do -> <E>does</E>`
- `nt -> <E>n't</E>`

Estas correcciones no nos parecen equivocadas pero no coinciden con lo esperado, por lo que buscamos la forma de mejorarlas. Esto se efectuó con dos cambios:

- Cuando un [token](#) de la versión corregida es idéntico al de la original pero a continuación aparece un [token](#) que forma parte de una contracción y representa un cambio respecto del original, se considera que el error comienza en el primer [token](#) de la contracción.
- Cuando se encuentra un error en un [token](#) y el que lo sigue es parte de una contracción, se considera que el [token](#) siguiente forma parte del error independientemente de si es un cambio respecto del original.

El resultado es que los cambios se marcan de la siguiente manera:

- `don't -> <E>doesn't</E>`
- `doesnt -> <E>doesn't</E>`

Por último, también hacemos una excepción para mejorar la detección de errores del tipo Artículo Faltante (MA). En el corpus original, estos errores suelen estar marcados alrededor del sintagma nominal donde falta el artículo, es decir:

`She wears pink t-shirt -> She wears <E_MA>a pink t-shirt</E_MA>`

Sin embargo, de acuerdo con las reglas que presentamos, el resultado que obtendríamos con nuestro sistema solo etiqueta el artículo agregado, dejando de lado el resto del sintagma. Para evitar esto, cuando se encuentra un artículo en el texto corregido que haya resultado de una inserción en el texto original, se marcan dentro del error todos los `tokens` siguientes hasta llegar a un sustantivo o nombre propio.

En resumen, esta parte del proceso recibe pares de textos donde el segundo es la versión corregida del primero. Tras utilizar un algoritmo de `WER` para alinearlos y seguir una serie de reglas para etiquetarlos, devuelve el segundo texto con las diferencias marcadas con etiquetas *dummy*. En el siguiente paso, estas etiquetas serán clasificadas entre los tipos de errores disponibles.

Tomando la misma oración del ejemplo visto en la sección 3.8.1, podemos ver el resultado luego de insertar las etiquetas:

```
<TBD> She </TBD> is <TBD> Andrea </TBD>.
<TBD> She is </TBD> 14 years old.
<TBD> She has </TBD> one dog.
```

3.9. Clasificador de errores

Una vez obtenidas las correcciones, debemos etiquetarlas con los tags de errores predefinidos. Para eso entrenamos un clasificador a partir de las correcciones halladas en el corpus.

Como se observa en la figura 3.2, entrenamos el clasificador a partir de pares de textos original - corrección. En nuestro caso, utilizamos únicamente las etiquetas disponibles en el corpus de desarrollo. En el corpus de validación se encuentran además otras etiquetas que no utilizamos en este proyecto, sin embargo, en trabajos futuros, sería posible entrenarlo con otro corpus y obtener etiquetas diferentes.

El proceso para obtener las correcciones del corpus es similar al que seguimos para identificar las mismas en las oraciones corregidas. En este caso disponemos de pares de oraciones donde una es el original con errores y otra es una versión corregida y etiquetada. Utilizamos el mismo algoritmo para alinear los pares de oraciones tomados del corpus, de forma de obtener un corpus de pares error-corrección, junto con el tipo del error.

Para alinear estas oraciones, en primer lugar consideramos las etiquetas del corpus como `tokens`, y las agregamos a nuestro diccionario. Luego, modificamos el algoritmo de forma que todas las operaciones debieran estar contenidas por un tag de apertura y su correspondiente tag de cierre. Esto es coherente con lo esperado ya que se supone que todos los cambios en las versiones corregidas están dentro de las etiquetas.

De esta forma obtuvimos un corpus de cerca de 300 errores, como se ve en la tabla 2.8, con su correspondiente corrección manual y el tipo del error.

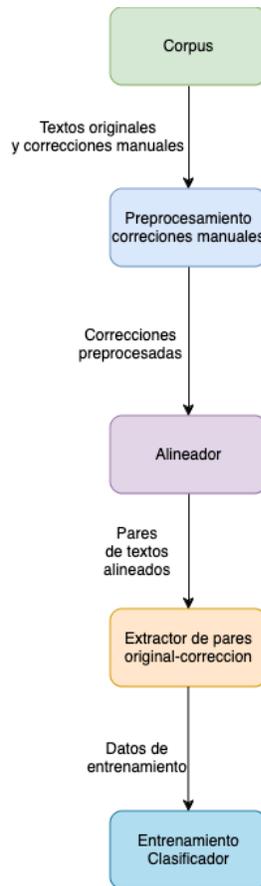


Figura 3.2: Diagrama del clasificador

3.9.1. Selección de características para el clasificador

Una vez completada esta tarea, procedimos a extraer las características adecuadas para el clasificador. Inicialmente, utilizamos [BERT](#) para obtener los [embeddings](#) del texto original y el texto corregido, utilizando la distancia entre ellos para intentar predecir el tipo de error. Sin embargo, este enfoque no produjo los resultados esperados. Por esta razón, optamos por seguir el enfoque presentado en el trabajo de [Swanson y Yamangil \(2012\)](#), cuyas características nos sirvieron de base para la selección de los atributos a evaluar.

Para calcular los atributos que requieren de información gramatical utilizamos la librería *spacy*⁴, que permite obtener la categoría gramatical (POS: Part of Speech) y el lema (es decir la forma base sin flexionar de un [token](#)) de las palabras correspondientes. Una consideración que se debe hacer es que las he-

⁴<https://spacy.io/>

ramientas de este tipo pueden tener limitaciones, y en particular en textos con errores como los que tenemos, es posible que no den siempre respuestas correctas. Por eso, incluimos también un atributo que indique si el **token** del texto original se encuentra en el diccionario. Para este atributo, utilizamos la lista de palabras disponible en la librería *nltk*⁵, buscando en la lista el lema del **token** obtenido utilizando *spacy*, ya que probamos distintas opciones y esta fue la que nos dio mejor resultado.

Las características utilizadas fueron las siguientes:

- Operación: el tipo de operación aplicada (inserción, sustitución o eliminación)
- POS del **token** anterior: *Part-of-Speech*⁶(POS) de la palabra anterior, si existe.
- POS del **token** siguiente: POS de la palabra siguiente, si existe.
- Prefijo común: la longitud del prefijo en común entre las palabras.
- ¿Mismo lema?: verifica si la palabra original y la corrección comparten el mismo lema (aplicable solo en casos de sustitución de una palabra).
- ¿Está en el diccionario?: indica si la palabra o frase original está registrada en el diccionario.
- Costo del cambio: utilizando el mismo algoritmo de WER ya presentado, pero, en este caso a nivel de caracteres en lugar de **tokens**, calculamos la distancia de edición entre el texto original y la corrección, solamente en el caso de sustituciones.
- POS de la corrección: el POS de la palabra corregida en casos de sustitución o inserción.

3.9.2. Entrenamiento del clasificador

Para el entrenamiento del clasificador, se utilizó la técnica de one-hot encoding, excepto en el caso de las características *Costo del cambio* y *Prefijo común*, que se trataron como valores continuos. El one-hot encoding es una técnica que transforma variables categóricas o discretas en un formato comprensible para los algoritmos de aprendizaje automático. Para cada valor único de una característica discreta, se crea una columna binaria(0 o 1), donde el valor correspondiente a una instancia se codifica como un 1 y todos los demás como 0.

En cuanto a la selección del modelo de clasificador a seguir, también nos basamos en el trabajo de Swanson y Yamangil (2012), eligiendo el clasificador MaxEnt, también conocido como regresión logística.

Este clasificador fue testeado de forma independiente al resto del sistema separando un 20 % de los textos del corpus de desarrollo.

⁵<https://www.nltk.org/>

⁶Categoría gramatical

Obtuvimos así un [accuracy](#) de 76% y un puntaje F_1 global de 76%. En la tabla 4.4 que se encuentra en la sección de experimentación, se detalla el F_1 por clase de error. Es importante notar que esto se alcanzó entrenando con apenas 220 errores que estaban en cerca de 30 textos.

Con estos valores de [accuracy](#) y F_1 relativamente altos, consideramos que el clasificador es satisfactorio para proceder con los experimentos utilizando este enfoque.

Una vez que completamos esta validación del clasificador, procedimos a entrenarlo nuevamente con el conjunto completo de errores obtenidos del corpus. Esta es la versión final que utilizamos para clasificar los errores hallados anteriormente que estaban marcados con etiquetas *dummy*.

Una vez clasificados los errores, la parte final del proceso es construir los textos etiquetados. De esta forma, para cada texto de la entrada se retorna un par de textos, el primero de ellos es la versión corregida por el [LLM](#) sin etiquetas, y la segunda es la versión etiquetada de esa misma corrección.

Capítulo 4

Experimentación

En esta sección, se analizarán y compararán los resultados obtenidos en distintos experimentos realizados con diversos LLMs y los resultados de nuestra propia solución, utilizando diferentes corpus de validación. En particular, se examinarán las diferencias de desempeño entre cada experimentación, destacando las mejoras alcanzadas y los desafíos encontrados en el proceso.

En todos los experimentos realizados en esta etapa se consideraron los tipos de errores descritos en la tabla 3.1. No son todos los tipos de errores que estaban en los corpus originalmente, ya que había errores poco recurrentes que descartamos, como se detalla en las secciones 3.6.1 y 4.3.

4.1. Métricas de corrección para el corpus del proyecto 2021 usando ChatGPT y LLaMA

Se llevó a cabo un análisis comparativo de la precisión de las correcciones realizadas por ChatGPT y LLaMA frente a los resultados obtenidos en Brown y Páez (2021), considerado como la referencia correcta. Este análisis se realizó utilizando el `prompt` final mencionado en la sección anterior.

Retomando la clasificación dada en el capítulo 2, que divide a los correctores en dos grupos, correctores de errores individuales y correctores de nivel oracional, podemos decir que los LLMs pertenecen al segundo grupo. Esto explica por qué aparecen correcciones de errores que no estaban en los casos de entrenamiento (A.3). Sin embargo, nuestro *gold standard* fue corregido concentrándose en los errores individuales y, además, que pertenecieran a ciertos tipos de errores.

Si consideramos el *gold standard* como un conjunto de correcciones perfectas y únicas, las correcciones que no estaban presentes en el proyecto anterior deberían ser consideradas como errores, es decir, serían falsos positivos. Sin embargo, observamos que la corrección en este caso no era exhaustiva y los criterios para identificar y marcar errores pueden ser ambiguos. Esto significa que dos correcciones pueden ser válidas a pesar de que no coincidan entre sí. De hecho, en Bryant y Ng (2015) se presenta un experimento donde se calcula el punta-

Original	She is andre hi is 14 years old
Corrección original	She is <E.C-PN> andre </E.C-PN> hi is 14 years old
Corrección ChatGPT	She is Andrea. She is 14 years old.
Etiquetas ChatGPT	<E.PN>: “hi” should be corrected to “She.”

Tabla 4.1: Corrección extra válida

je $F_{0,5}$ de comparar correcciones hechas por humanos contra un *gold standard*. El puntaje promedio obtenido por correctores humanos, comparando contra un *gold standard* hecho a partir del trabajo de nueve anotadores es de 72.58. En el caso de un *gold standard* con correcciones de dos anotadores (lo más parecido a nuestro caso, ya que tenemos las anotaciones del proyecto anterior y nuestra propia intuición), el $F_{0,5}$ promedio obtenido por otros anotadores es de 56.68.

Esos resultados no se deben a un mal trabajo de los anotadores humanos, sino a la discrepancia esperada en tareas como la *GEC*, y son esperables. Es decir que no siempre es adecuado considerar todas las correcciones distintas a las disponibles en el corpus como erróneas. Para evitar penalizar indebidamente al sistema, consideramos una categoría más que corresponde a los errores correctamente identificados por el modelo, pero que no estaban marcados en el *gold standard*. El agregado de esta categoría reduce los casos que originalmente habrían sido considerados como FP, separándolos entre los que están correctamente corregidos y los que o están mal corregidos o era innecesario hacerlo.

Es importante notar que en el caso de las correcciones etiquetadas, consideramos que los errores marcados pertenecían a esta categoría aun cuando las etiquetas no estuvieran bien, ya que naturalmente el sistema solo dispone del conjunto de etiquetas usado originalmente.

Por ejemplo, en la tabla 4.1, se puede observar un ejemplo tomado del corpus. En este caso, el cambio *hi* por *She* no estaba presente en la corrección del proyecto anterior pero sí fue hecha por el modelo. Sin embargo, la etiqueta asignada fue “E.PN”, la cual no corresponde con el tipo de error. El problema es que este cambio no se corresponde con ninguno de los tipos de errores definidos, por lo que no hay una etiqueta aceptable, por eso lo consideramos como una corrección válida.

La única excepción a esto es en algunos casos donde el sistema hace correcciones que sí pertenecen a las categorías dadas (y que faltaban en el *gold standard*), en ese caso solo la consideramos como correcta si el etiquetado era correcto, por ejemplo el caso mostrado en la tabla 4.2

La nueva categoría no afecta directamente a los puntajes obtenidos por el sistema ya que no se utiliza en el cálculo de *precision*, *recall* o $F_{0,5}$. Sin embargo, los afecta indirectamente pues, como dijimos, se reduce la cantidad de FP. Consideramos que esto es aceptable como un camino intermedio ya que, si bien no penalizamos estos casos como FP, tampoco los consideramos dentro de los casos positivos.

A futuro, podría ser útil, para mejorar la calidad del *gold standard*, agregar los errores “nuevos” que sean validados, especialmente aquellos pertenecientes

Original	she like pizza but don't apple
Corrección anterior	<E_C-BOS> she </E_C-BOS> <E_SVA> like </E_SVA> pizza but don't <E_MA> apple </E_MA>
Corrección ChatGPT	She likes pizza but doesn't like apples.
Etiquetas ChatGPT	<E_PN>: "don't" should be corrected to "doesn't" to agree with the subject "she".

Tabla 4.2: Corrección extra mal etiquetada

a las categorías de errores que ya están disponibles (como el caso presentado en la tabla 4.2. Para agregar errores como el presentado en la tabla 4.1 sería necesario además definir y agregar nuevos tipos de error. Esto también sería una mejora posible para el [gold standard](#), pero requiere un mayor trabajo. De todas formas, creemos que a futuro sería importante revisar qué tipo de correctores son mejores para este tipo de tarea, y, en consecuencia, cómo se debe medir la eficacia de los correctores. Hablamos más sobre este tema en la sección 5.3.4.

4.2. Pruebas realizadas con ChatGPT y LLaMA

Para llevar a cabo esta evaluación, se diseñaron pruebas empleando dos enfoques distintos para ChatGPT y LLaMA: una configuración sin etiquetas y otra con etiquetas.

El dataset utilizado estaba equilibrado en cuanto a la distribución de ejemplos según la puntuación. Los resultados se recopilaron obteniendo los siguientes campos:

- ID
- Texto original
- Corrección proyecto anterior
- Corrección ChatGPT
- Corrección con Etiquetas ChatGPT
- Errores totales
- True positives
- False negatives
- False positives
- Correcciones extra
- Precision
- Recall

- $F_{0,5}$
- True positives(Corrección con Etiquetas)
- False negatives(Corrección con Etiquetas)
- False positives (Corrección con Etiquetas)
- Correcciones extra (Corrección con Etiquetas)
- Precisión (Corrección con Etiquetas)
- Recall (Corrección con Etiquetas)
- $F_{0,5}$ (Corrección con Etiquetas)

Cabe destacar que las **Correcciones extra** se refieren a errores correctamente identificados y corregidos por estos modelos, pero que no habían sido detectados previamente por el corrector basado en reglas del proyecto anterior. Es importante destacar que todos los cambios “extra” realizados por los modelos fueron analizados caso a caso, y solo los agregamos a esta columna si eran correcciones válidas y necesarias. No aceptamos cambios que fueran equivocados o que cambiaran una expresión correcta por otra también correcta. Todos esos casos los consideramos, como sería lo habitual, como false positives.

A modo de ejemplo ilustrativo, examinemos la frase original presentada por el estudiante y su correspondiente corrección según el método empleado en el proyecto anterior.

A continuación presentaremos un texto del corpus, con tres conjuntos de correcciones: en primer lugar, las correcciones del proyecto anterior basado en reglas, que presentamos en la sección 2.2.2, y luego los resultados de aplicar los dos métodos de corrección con ChatGPT: con y sin etiquetas. Para cada caso analizaremos las diferencias entre los errores encontrados.

Frase original:

she birhtday 14 years old i like pizza,right and ried and dont like, single,apple and fishing.

Corrección del proyecto basado en reglas:

<E_C-BOS>she</E_C-BOS> <E_S>birhtday</E_S> 14 years old <E_C-PI>i</E_C-PI> like pizza,right and <E_S>ried</E_S> and dont like,single,<E_MA>apple</E_MA> and fishing.

Corrección de ChatGPT sin Etiquetas:

She is 14 years old. I like pizza, right, and rice,
but I don't like being single, apples, or fishing.

Al comparar la corrección proporcionada por el proyecto anterior con los resultados obtenidos de ChatGPT, se pueden observar los siguientes aspectos:

- True Positives (TP): Los elementos corregidos correctamente por ambos métodos se destacan en verde.
- False Negatives (FN): Los errores que no fueron detectados por ChatGPT se indican en rojo.
- False Positives (FP): En este ejemplo específico, no se observaron errores extra mal (false positives).
- True Negative(TN): Las correcciones adicionales realizadas correctamente por ChatGPT se marcan en azul.

She is 14 years old. I like pizza, right, and rice, but I don't like being single,
apples, or fishing.

Corrección ChatGPT con Etiquetas:

Un detalle a considerar cuando trabajamos con ChatGPT y LLaMA es que al pedirle las etiquetas para las correcciones efectuadas, retornan una lista de las correcciones, y no el texto con las etiquetas en sus posiciones correspondientes. A continuación mostramos la respuesta dada por ChatGPT.

```
<E_S>: "birhtday" should be "birthday."  
<E_BOS>: "she" should be capitalized at the beginning of the sentence.  
<E_PN>: "i" should be capitalized to "I."  
<E_UA>: "single" should not have an article "a" before it.  
<E_MA>: "the" is missing before "rice."  
<E_VF>: "ried" should be corrected to "rice."  
<E_SVA>: The verb form "dont" should be corrected to "don't."  
<E_UA>: "single" should not have an article "a" before it.  
<E_MA>: "the" is missing before "apple."  
<E_UA>: "fishing" should not have an article "a" before it.
```

Al aplicar el mismo proceso de comparación contra el proyecto basado en reglas, se observa que los siguientes errores fueron detectados correctamente:

```
<E_S>: "birhtday" should be "birthday."  
<E_BOS>: "she" should be capitalized at the beginning of the sentence.  
<E_MA>: "the" is missing before "apple."
```

En contraste, el resto de las correcciones identificadas por ChatGPT no eran correctas, resultando en tres true positives y siete false negatives.

Siguiendo el proceso del ejemplo, se calcularon las métricas de [precision](#), [recall](#) y $F_{0,5}$ para ejemplos seleccionados.

Con el fin de ofrecer una visión general de los resultados obtenidos, se realizó un cálculo de los promedios para las columnas de Correcciones extra, Precisión, Recall y $F_{0,5}$, considerando ambos enfoques: correcciones de ChatGPT con y sin etiquetas. Luego, aplicamos el mismo proceso a las correcciones realizadas por LLaMA.

Enfoque	Correcciones extra	Precision	Recall	$F_{0,5}$
ChatGPT sin etiquetas	1.512	0.851	0.873	0.848
LLaMA sin etiquetas	0.860	0.900	0.843	0.880
ChatGPT con etiquetas	1.070	0.514	0.460	0.460
LLaMA con etiquetas	0.581	0.588	0.422	0.508

Tabla 4.3: Resultados utilizando ChatGPT y LLaMA

Se puede observar en la tabla 4.3 que el número de correcciones extra hechas por ChatGPT y LLaMA sin Etiquetas es de aproximadamente 1.5 y 0.9 respectivamente, lo que sugiere que ambos modelos poseen una mayor capacidad de corrección en comparación con el algoritmo utilizado en el proyecto anterior. Este conteo fue hecho de forma manual y solo se consideraron las correcciones que fueran válidas y necesarias. Este fenómeno puede explicarse por varias razones:

- A diferencia de algoritmos tradicionales, que pueden estar limitados a reglas predefinidas o patrones de errores conocidos, ChatGPT y LLaMA tienen la capacidad de generalizar a partir de grandes cantidades de datos de entrenamiento. Esta habilidad les permite aplicar correcciones incluso en situaciones no vistas previamente, aumentando así la probabilidad de corregir errores adicionales que el algoritmo anterior no habría identificado.
- Los modelos como ChatGPT y LLaMA tienen la capacidad de comprender el contexto de manera más profunda y detallada, lo que les permite no solo corregir errores evidentes, sino también hacer ajustes sutiles que mejoran la calidad general del texto. Esta comprensión contextual es fundamental para identificar errores extra o errores que no son triviales, lo que puede explicar el mayor número de correcciones adicionales en comparación con el algoritmo utilizado en el proyecto anterior. Como mencionamos en la sección 5.3.4, a futuro sería importante considerar si se quiere utilizar este tipo de correctores para corregir textos de niveles iniciales, o si sería mejor corregir sólo errores pertenecientes a un conjunto acotado.

Además de analizar valores puntuales que muestran similitudes, resulta interesante realizar una comparación entre aquellas tablas que presentan diferencias significativas, es posible identificar diversas razones que podrían justificar las variaciones observadas en los valores de [precision](#), [recall](#), $F_{0,5}$, y en la cantidad de correcciones extra entre los distintos enfoques y modelos evaluados.

En el caso de la corrección con etiquetado, se puede ver fácilmente que se obtienen valores de [precision](#), [recall](#) y $F_{0,5}$ mucho peores que en la corrección sin etiquetas, tanto en el caso de ChatGPT como en el de LLaMA. La diferencia entre estos dos enfoques, es que en el caso sin etiquetas, no se le pide al modelo que “comprenda” los cambios hechos. En cambio en el enfoque con etiquetas es necesario que el modelo pueda dar algo de información sobre las correcciones: qué tipo de error y cuál fue la corrección, además de identificar correctamente el lugar donde está el cambio. Dado que se logra mucho mejor la primera tarea, podemos decir que estos modelos son razonablemente buenos para identificar y corregir errores, pero son malos para identificar de qué tipo de error se trata y qué correcciones fueron hechas. Esto puede deberse a varias razones:

- Un diseño ineficaz de las etiquetas en el [prompt](#).
- Una complejidad adicional que el modelo no es capaz de manejar eficientemente.
- La existencia de diferentes correcciones posibles, lo que causa que algunas etiquetas se vuelvan menos sistemáticas y predecibles
- Los resultados podrían también indicar un sesgo en los algoritmos de corrección hacia ciertos tipos de errores o estructuras lingüísticas.

Si bien no profundizamos más en este aspecto, consideramos que, dado que estos modelos no son entrenados con textos con errores y mucho menos con textos con errores etiquetados (y aún menos con las etiquetas utilizadas en este trabajo), es improbable que el texto etiquetado que generen sea adecuado. Aunque discutimos la posibilidad de llevar a cabo un fine-tuning del modelo para incorporar información de las etiquetas, consideramos que el tamaño del corpus del que disponemos no es suficiente para lograr un entrenamiento bueno, y a eso se le suma la dificultad para acceder a computadoras con los recursos necesarios para llevarlo a cabo. Por esto, finalmente decidimos abandonar esa idea.

4.3. Pruebas realizadas con nuestra solución

Como presentamos en la sección [3.4](#), implementamos una aplicación que recibe pares de textos, uno con errores escrito por estudiantes y su corrección etiquetada. Estos textos son utilizados para entrenar un clasificador, y recibe a su vez un conjunto de textos escritos por estudiantes (que puede coincidir o no con el conjunto antes mencionado), los corrige utilizando un [LLM](#) y los etiqueta utilizando el clasificador entrenado.

4.3.1. Pruebas del clasificador aislado

Como reportamos en la sección [3.9](#), probamos de forma individual el clasificador de errores. Esto fue hecho dividiendo el corpus de desarrollo y utilizando

el 80 % como corpus de entrenamiento y el restante 20 % como corpus de validación.

Realizamos dos experimentos para este módulo, utilizando los mismos corpus de entrenamiento y validación. En el primero de ellos, utilizamos features tradicionales para entrenar al modelo y en el otro utilizamos `embeddings` calculados para el texto original y su corrección.

El resultado global obtenido por el clasificador con las features presentadas fue de 0.76. Si observamos el puntaje obtenido por tipo de error, vemos que los casos de mayúsculas faltantes son resueltos siempre de forma correcta, lo cual es esperable dado que son casos “fáciles” de identificar. Por debajo están los errores de concordancia y ortografía, que igualmente son relativamente buenos.

A continuación aparecen los errores de forma del verbo. En nuestro conteo manual de errores observamos que en algunos casos se confundían con errores de concordancia y en otros con errores de ortografía. Creemos que esto es esperable ya que la etiqueta es bastante ambigua en el sentido de que no indica muy precisamente es el cambio hecho o en qué condiciones debe hacerse. Lo más común en los textos disponibles, por su naturaleza y temática, es el cambio de infinitivo a gerundio y viceversa, pero incluso estos dos tipos no necesariamente se parecen en cuanto a las features que elegimos.

Por último están los errores de artículos, donde el caso de artículo faltante es mucho peor que el de artículo innecesario. Creemos que este tipo de errores es difícil de predecir ya que observamos que no solo se etiqueta al artículo que se agrega o se elimina, sino a todo el sintagma nominal. Creemos que esto hace que sea más difícil detectar esos casos y que puede ser una razón por la que el puntaje de esta clase de errores fue tan bajo.

En cuanto al modelo entrenado con `embeddings`, obtuvimos un puntaje mejor al obtenido por las *features* usadas anteriormente. Lamentablemente, este experimento fue hecho sobre el final del proyecto, y por falta de tiempo, no pudimos volver a evaluar los resultados de corregir los corpus que teníamos utilizando este clasificador. Creemos que los resultados globales deberían mejorar en concordancia con las mejoras en el clasificador, pero esto debería ser explorado en un trabajo futuro. En la tabla 4.4 presentamos el los resultados por categoría de error y global para ambos experimentos.

Etiqueta	F ₁ con features	F ₁ usando <code>embeddings</code>
E_C-BOS	1.0	0.97
E_C-PI	1.0	1.0
E_C-PN	1.0	0.8
E_MA	0.2	0.86
E_S	0.82	0.86
E_SVA	0.93	0.77
E_UA	0.5	1.0
E_VF	0.58	0.86
Global	0.76	0.88

Tabla 4.4: Puntaje F₁ por etiqueta

4.3.2. Pruebas de la solución completa

Para la solución completa realizamos dos pruebas, 2.2.2, una con el corpus de desarrollo, que también habíamos utilizado para entrenar el clasificador, y luego el corpus de test que hasta entonces no habíamos utilizado.

Es importante notar que en la etapa de corrección con un LLM, tanto el corpus de desarrollo como el de test son “nuevos” para el modelo, ya que el modelo está pre-entrenado y no hicimos ningún tipo de fine-tuning, y el prompt (A.2.6) utiliza tres textos que no pertenecen a ninguno de los dos corpus. Además, los textos corregidos que son generados por el LLM, que luego recibe el clasificador, no coinciden en ningún caso con los que fueron utilizados para entrenarlo. De todas formas, la evaluación con el conjunto de test es totalmente independiente de forma de garantizar que su desempeño no estuviera sesgado por los datos de entrenamiento y para validar su capacidad de generalización en un conjunto de datos nuevo.

Con el fin de ofrecer una comparación más directa con la sección 4.1, se presenta la tabla 4.5, donde presentamos los resultados con el mismo formato que en esa sección. Es decir, precision, recall y $F_{0,5}$, junto con el promedio de correcciones extra halladas por el modelo, que no estaban en el gold standard. Los datos reportados para Mistral sin etiquetas en el corpus de desarrollo son comparables con los enfoques sin etiquetas de la sección anterior. Lo presentamos a modo de baseline para la solución generada, ya que por lo expuesto en la sección 2.3.2 no pudimos utilizar ni ChatGPT ni LLaMA en nuestra solución. Los datos reportados para nuestra solución son comparables con los resultados con etiquetas de la sección anterior.

Enfoque	Corpus	Correcciones extra	Precision	Recall	$F_{0,5}$
Mistral sin etiquetas	Desarrollo	1.698	0.815	0.820	0.804
Solución	Desarrollo	1.643	0.726	0.511	0.609
Mistral sin etiquetas	Test	1.125	0.873	0.725	0.813
Solución	Test	1.100	0.782	0.549	0.682

Tabla 4.5: Resultados de nuestra solución

Observando la tabla 4.5, se ve que las métricas de precision, recall y $F_{0,5}$ obtenidas por Mistral son consistentemente inferiores en comparación con las generadas por ChatGPT y a LLaMA (presentados en la tabla 4.1). Este resultado era previsible, dado que ambos modelos, al ser más avanzados, tienden a ofrecer un rendimiento superior en tareas de procesamiento de lenguaje natural en comparación con Mistral 7B. Sin embargo, como explicamos en la sección 2.3.2, no es posible utilizar ChatGPT de forma libre, y no pudimos ejecutar el modelo LLaMA 2 70B que utilizamos para las pruebas de forma local. Considerando que Mistral es un modelo gratuito y de código abierto, que fue posible ejecutarlo localmente, y que los puntajes obtenidos por este son relativamente altos, creemos que es una alternativa viable en nuestro contexto. De todas formas, la aplicación que generamos es bastante modular y sería posible modificar el módulo del LLM para que utilizara cualquier otro modelo similar sin

modificar el resto de la solución.

Comparando los resultados presentados en la tabla 4.5 con los disponibles en la tabla 4.3 en los casos con etiquetas, se observa un incremento de 10% con respecto a LLaMA y de 15% con respecto a ChatGPT en $F_{0,5}$ al utilizar nuestra solución. Este resultado es especialmente positivo si consideramos que los resultados sin etiquetas utilizando Mistral son peores que los resultados de ChatGPT y LLaMA.

Pasando al corpus de test, se puede observar que los resultados obtenidos por Mistral sin etiquetas son similares a los obtenidos en el corpus de desarrollo, lo cual es esperable ya que este modelo está pre-entrenado.

Es importante mencionar que, para evaluar los resultados en el corpus de test, no consideramos las etiquetas como “otro error” (Other Error, OE), “sujeto faltante” (Missing Subject, MS), “verbo faltante” (Missing Verb, MV), “preposición incorrecta” (Wrong Preposition, WP), “sujeto incorrecto” (Wrong Subject, WS), “verbo incorrecto” (Wrong Verb, WV), “orden incorrecto de las palabras” (Incorrect Word Order, IWO), “números de sustantivos” (Noun Numbers, NN), “pronombre” (PR) y “preposición faltante” (Missing Preposition, MPT). Consideramos estas correcciones como true positives si la corrección realizada por Mistral es correcta, independientemente de la etiqueta del tipo de error predicha. Dado que nuestra solución no fue entrenada con estas categorías, es imposible que pudiera generarlas. Además tampoco consideramos las etiquetas de “puntuación” (P) ya que nuestro algoritmo postprocesa los errores de puntuación sin asignarles etiquetas.

En cuanto a aplicar nuestra solución al corpus de test, se obtuvo un mejor resultado en todas las métricas consideradas, con un $F_{0,5}$ 7% más alto que en el corpus de desarrollo. Esto podría deberse a las consideraciones mencionadas anteriormente, ya que, si realmente incluyéramos todas esas etiquetas como parte del entrenamiento, es posible que no todas ellas sean consideradas como verdaderos positivos, aunque ahora continúan siendo clasificados como tales.

Sin embargo, también es importante señalar que en el corpus hay correcciones de errores de las siguientes formas:

- “punk t-shirt” → “pink t-shirt”
- “shirt” → “t-shirt”
- “he” → “she”

Estas correcciones están fundamentadas en el contexto de la consigna (color de la camiseta, tipo de prenda, y el hecho de que Andrea es mujer). No obstante, es crucial destacar que estas partes de algunas oraciones de los estudiantes no presentan errores gramaticales sino que solo se pueden catalogar como errores conociendo el contexto del ejercicio.

Adicionalmente, calculamos cada ocurrencia de error como una entidad individual en lugar de hacerlo por oraciones, obteniendo los siguientes resultados.

Corpus	Corrección	Precision	Recall	F _{0,5}
Desarrollo	Sin etiquetas	0.915	0.862	0.904
Desarrollo	Con etiquetas	0.741	0.451	0.656
Test	Sin etiquetas	0.890	0.752	0.858
Test	Con etiquetas	0.817	0.536	0.740

Tabla 4.6: Promedios de las correcciones de Mistral y nuestra solución

Se puede observar que las métricas de [precision](#), [recall](#) y $F_{0,5}$ aumentaron consistentemente bajo las mismas condiciones de prueba. Esta diferencia puede explicarse porque los textos con puntajes muy bajos (incluso con puntaje 0) suelen ser textos cortos con pocos errores marcados. Al tomar los textos como unidades de análisis, estos puntajes bajos impactan de manera más significativa los resultados promediados, lo que disminuye las métricas globales. Al cambiar el enfoque y calcular por ocurrencia de error, el impacto de las métricas bajas en oraciones específicas se diluye.

Dificultades encontradas

Observamos algunas inconsistencias en el etiquetado de ciertos errores, por ejemplo, en los casos de la etiqueta MA combinada con NN, y, por otro lado, de SVA con VF, que se muestran a continuación.

En el caso de MA con NN, consideremos las siguientes oraciones:

original	corregida
she like pizza but don't apple	<E_C-BOS>She</E_C-BOS> <E_SVA>likes</E_SVA> pizza but don't <E_MA>apples</E_MA>
don't like apple	<E_MS>she</E_MS> <E_SVA>doesn't</E_SVA> like <E_NN>apples</E_NN>

Tabla 4.7: Ejemplo de diferentes correcciones de MA y NN

Aunque ambos ejemplos requieren cambiar *apple* por *apples*, las etiquetas aplicadas en cada caso son distintas.

En el caso de SVA con VF, el ejemplo en la tabla 4.11 lo ilustra:

original	corregida
she like riding a bike	<E_C-BOS>She</E_C-BOS> <E_VF>likes</E_VF>riding a bike
she like riding a bike	<E_C-BOS>She</E_C-BOS> <E_SVA>likes</E_SVA> riding a bike

Tabla 4.8: Ejemplo de diferentes correcciones de SVA y VF

Dado lo anterior, consideramos que si nuestra predicción identifica tanto VF como SVA, ambos deben contarse como correctos si la corrección corresponde a un caso de concordancia entre sujeto y verbo.

Otro problema interesante que encontramos durante el desarrollo es que puede haber más de una solución posible para las correcciones. Para ilustrar este problema, citamos algunos ejemplos del artículo (Felice y Briscoe, 2015) y también proporcionamos ejemplos propios.

Source	Annotator 1	Annotator 2
This machines is designed for help people.	(This → These), (is → are), (help → helping)	(machines → machine), (for → to)

Tabla 4.9: Source sentence and annotations

System hypothesis	System edits	P	R	F _{0,5}
This machines is designed for help people.	(This → These), (is → are), (for → to)	0.67	0.67	0.67

Tabla 4.10: Source sentence and annotations

En el artículo se presenta un texto corregido por dos anotadores, y el sistema propone una solución intermedia entre las correcciones de los dos. Se observa que el sistema proporciona una corrección válida, pero como la comparación se hace con el Anotador 1 o el Anotador 2, el mejor resultado se obtiene al comparar con el Anotador 1, alcanzando una [precision](#) de 0.67, un [recall](#) de 0.67 y un F_{0,5} de 0.67. En el caso de comparar con el Anotador 2, los resultados serían peores.

En nuestros ejemplos también se encuentran dificultades similares. La [tabla 4.11](#) ilustra este problema:

Original	Corrección manual	Mistral
but she don't like ate apples.	but she <E_SVA>doesn't </E_SVA>like <E_VF>to eat </E_VF>apples.	but she doesn't like eating apples.

Tabla 4.11: Ejemplo de diferentes correcciones de SVA y VF

Como se observa en la tabla, Mistral realizó un cambio diferente en la corrección de *ate*, modificándolo a *eating* en lugar de *to eat*. Ambos cambios son válidos. En nuestro proyecto, consideramos correcto el cambio si coincide con las correcciones del [gold standard](#) en cada caso: las correcciones hechas por graduandos o por tutores.

Capítulo 5

Conclusiones y Trabajo Futuro

A continuación presentamos los resultados finales del proyecto así como un análisis de las dificultades que encontramos al llevarlo a cabo. Finalmente, presentamos posibles direcciones para continuar explorando el campo de [GED](#) y [GEC](#), y cómo se podrían utilizar [LLMs](#) en este sentido.

5.1. Resultados finales

Analizando la solución generada, consideramos que el resultado fue satisfactorio. El sistema presentado consiste en tres partes independientes que funcionan bien de forma independiente y también globalmente:

- El corrector que utiliza Mistral, con un $F_{0,5}$ de 0.80.
- El clasificador de errores, con un F_1 de 0.76. En este caso hallamos una opción que de hecho lo superaba pero no hubo tiempo para probarla en el sistema completo.
- El marcador de errores, que no evaluamos de forma individual.
- El $F_{0,5}$ del sistema completo considerando la corrección, el etiquetado y la identificación de errores fue de 0.60.

El objetivo principal de este proyecto era la creación de un herramienta que se pudiera utilizar para corregir textos escritos por estudiantes de inglés de niveles iniciales. La herramienta debía recibir estos textos y retornar una versión corregida de ellos, donde se marcaran los errores hallados. Como hemos presentado, generamos una solución que, a través del uso de modelos neuronales como los [LLMs](#), realiza exactamente este trabajo, y que lo hace razonablemente bien. En este sentido, consideramos que el objetivo principal ha sido cumplido.

Ahora bien, lo que buscamos inicialmente era llevar a cabo esta tarea utilizando de principio a fin LLMs. Como reportamos en la sección 3.3, esto no fue posible. No hallamos una forma de utilizar estas herramientas para generar los textos con los errores marcados de forma satisfactoria. Esto nos llevó a implementar la solución tal y como la presentamos, lo que de todas formas cumple con el objetivo, y, de hecho, no es un enfoque que hayamos visto en la literatura, y consideramos que vale la pena investigar.

Otros objetivos de este proyecto incluían justamente la investigación de los distintos enfoques utilizados para realizar esta tarea, principalmente considerando las distintas competencias realizadas en el área de GED y GEC. Como observamos, la única competencia en la que se utilizaron LLMs es la más reciente (y en ese caso, solo para detectar errores), e incluso en esa, no se utilizaron los modelos más modernos que utilizamos nosotros. Sin embargo, muchos de los participantes en esas competencias han publicado otras investigaciones dentro de esta área, las cuales tomamos como referencia para nuestro trabajo en el proyecto.

Respecto a los modelos utilizados, nuestra decisión de probar inicialmente con ChatGPT y LLaMA, y después con Mistral se debió más a cuestiones técnicas o de acceso a los modelos que al análisis que hayamos podido hacer sobre ellos. Esto es un deber dentro del proyecto ya que existen muchos otros modelos que deberían ser probados en esta tarea. De todas formas es importante decir que creemos que no encontramos una forma de hacer que las pruebas de estos modelos puedan ser automatizadas. Consideramos que esto dificulta mucho esa tarea.

El último objetivo se relaciona principalmente con la primera parte del proyecto, ya que se buscaba experimentar con LLMs para resolver la tarea con técnicas de *prompting*. Como se muestra en la sección 3.3, probamos distintas versiones de un *prompt* para lograr esto pero no encontramos una forma de obtener los textos con correcciones marcadas satisfactoriamente. En cuanto a la corrección a secas, sin marcar y etiquetar errores, esto lo hicimos simplemente generando un *prompt* con tres ejemplos tomados del corpus, pero no consideramos que el resultado se pueda mejorar mucho cambiando este *prompt*.

En resumen, creemos que los objetivos se han cumplido en términos generales y que el objetivo principal se cumplió satisfactoriamente, pero aún queda más trabajo por hacer para mejorar el desempeño de este sistema u otros similares.

5.2. Dificultades encontradas

5.2.1. Falta de conocimiento previo

El primer desafío que encontramos fue nuestra falta de conocimiento previo sobre el funcionamiento de los LLMs. No contábamos con la base teórica necesaria para abordar estos temas en profundidad.

Para superar esta dificultad, dedicamos tiempo adicional a investigar y estudiar por nuestra cuenta. Exploramos la literatura académica, artículos especia-

lizados y documentación técnica para entender mejor los conceptos claves. Este fase de aprendizaje fue importante para poder enfrentar de manera adecuada los desafíos relacionados con [LLMs](#) y su aplicación en nuestro proyecto.

5.2.2. Limitaciones técnicas en la ejecución de modelos de lenguaje

Durante el proceso, nos encontramos con limitaciones técnicas al intentar ejecutar modelos de lenguaje de gran tamaño ([LLMs](#)) de manera local. Realizamos pruebas iniciales utilizando ChatGPT de forma de tener una baseline para comparar, pero lo descartamos para la creación de una solución debido a que no es de uso libre, mientras que LLaMA, aunque accesible, requiere una gran capacidad de procesamiento, particularmente de GPUs, lo que dificultó su implementación. A pesar de contar con acceso a ClusterUY, solo se podía ejecutar LLaMA en las tres computadoras con GPUs adecuadas, lo que limitó nuestra capacidad de trabajo.

5.2.3. Desafíos derivados del corpus

Otra de las dificultades que encontramos se presentó durante la fase de experimentación. Existe una diferencia significativa entre el corpus de desarrollo y el de test en cuanto a la cantidad y tipo de errores abordados en la corrección de los textos estudiantiles. Esta disparidad complicó el proceso cuando validamos con el corpus de test, ya que tuvimos que aplicar algunos criterios propios para realizar las correcciones manuales.

Como autocrítica, y tal como se mencionó previamente, no seleccionamos los errores de tipo MS, MV y OE para el entrenamiento del clasificador debido a la escasez de ejemplos. Sin embargo, podríamos haber utilizado el corpus de test como conjunto de entrenamiento, ya que este presenta una mejor distribución de los distintos tipos de errores. Una posible mejora sería entrenar el clasificador utilizando este corpus y validar los resultados con el que fue nuestro corpus de desarrollo, es decir a la inversa de lo que realmente hicimos.

Como mencionamos en la sección [4.3.2](#), al momento de determinar si las correcciones efectuados por los modelos eran correctas, encontramos algunas dificultades relacionadas con las etiquetas disponibles en el corpus, y también dificultades relacionadas con cómo se deben puntuar correcciones que concuerdan parcialmente con el corpus. Si bien estas dificultades no imposibilitan que se complete la tarea, si afectan en parte los resultados finales.

5.3. Trabajo futuro

La solución que proponemos es un sistema relativamente modular. Creemos que en general mejorar el funcionamiento de uno de sus partes tendería a mejorar el funcionamiento del sistema en su conjunto. De esta manera, podemos agrupar el trabajo futuro por módulo.

5.3.1. Corrección de textos

En primer lugar, consideramos que la corrección de los textos por LLMs será cada vez mejor en el futuro. Como ya mencionamos antes, desde el comienzo de este proyecto hasta el momento de escribir este informe, todos los servicios que utilizamos han presentado versiones más nuevas. De hecho, el modelo que utilizamos en la solución, Mistral 7B, fue lanzado durante el curso del proyecto. Este campo está en pleno desarrollo y el área de GEC y GED es de interés por lo que creemos que en la medida que aparezcan nuevos modelos y nuevas versiones de los existentes, tenderán a mejorar en este aspecto.

Sin embargo, la existencia de modelos mejorados no es suficiente. Como dijimos, estos modelos requieren muchos recursos para ser ejecutados y, si bien es posible que haya optimizaciones en su desempeño y se vuelvan más accesibles en este sentido, hoy en día ejecutar los modelos más potentes es un desafío. Si se quiere mejorar en este aspecto, hoy en día parece que la única opción es destinar recursos económicos para mejorar la infraestructura de forma que se puedan ejecutar modelos más potentes.

Esta pieza del sistema, además, es la que toma más tiempo de ejecución, por lo que si se quisiera utilizar en un caso de uso en tiempo real, sería fundamental proveer los recursos para que tome un tiempo adecuado.

De todas formas, y como última consideración en este tema, se puede observar que los puntajes obtenidos solamente en corrección, sin etiquetado, por los tres modelos utilizados supera el 80%. Es decir que estos modelos ya hacen un trabajo bastante bueno, aunque, repetimos, ejecutarlos de forma libre y rápida no es tan directo al día de hoy.

5.3.2. Marcado de textos corregidos

La segunda parte del sistema es el marcado de los textos. Aquí es donde creemos que se pueden hacer mejoras más claras ya que, como dijimos, en el corpus que tenemos disponibles (y más generalmente, en los textos escritos por estudiantes de nivel inicial), aparecen en general muchos errores contiguos, y la heurística que tomamos no nos permite separarlos. Aunque no hicimos experimentos con respecto a esto, no creemos que una heurística simple como dejar todos los cambios como errores individuales mejore el resultado. Creemos que un sistema de reglas más complejo o un clasificador como el presentado por Xue y Hwa (2014) seguramente mejoren esta parte.

5.3.3. Clasificación de errores

La parte final de la solución es el clasificador de errores. Como ya mencionamos, un clasificador entrenado con *word embeddings* en lugar de *features* tradicionales tuvo un mejor puntaje cuando lo probamos aisladamente. Creemos que utilizar este clasificador mejoraría el resultado global.

Por otro lado, el conjunto de categorías que utilizamos es limitado. No cubre el conjunto de errores posibles (esto sería una exigencia demasiado alta), pero

tampoco cuenta con una etiqueta para “otros errores” que englobe lo que no es cubierto por las otras etiquetas. Consideramos que una mejora posible sería agrandar el conjunto de etiquetas para cubrir una mayor cantidad de casos de error.

Además de esto, como mencionamos en el caso de SVA y VF, parecería que ciertas categorías pueden confundirse o solaparse. Consideramos que sería interesante encontrar conjuntos de categorías que faciliten o promuevan la tarea de clasificación para estos textos. O, alternativamente, se podrían analizar conjuntos de etiquetas más estandarizadas o que sean utilizadas en otros corpus. De esta forma, sería posible entrenar el clasificador con textos que no necesariamente provengan del corpus utilizado por nosotros.

En relación a este último punto, la poca cantidad de ejemplos de entrenamiento afecta la calidad del clasificador. Sería positivo disponer de más textos anotados, ya sea del corpus existente ahora o de otros corpus, si se considera la posibilidad de cambiar las etiquetas utilizadas.

5.3.4. Consideración sobre los tipos de correctores

Una consideración al respecto de este tema es que el trabajo previo que presentamos, tanto en las competencias como en las anotaciones del corpus utilizado, se encuentra dentro del campo de los correctores de errores individuales. Sin embargo, de acuerdo con [Sakaguchi, Napoles, Post, y Tetreault \(2016\)](#), este tipo de correctores generan oraciones que no suenan naturales a los oídos de los hablantes nativos.

Lo que se debe observar es que los antecedentes presentados, salvo la última competencia, MultiGED-2023, son todos anteriores al advenimiento de los [LLMs](#) que utilizamos en este proyecto. Previo a estos modelos, no había herramientas que fueran muy buenas para reescribir textos enteros de forma que sonaran naturales además de corregir errores ([Östling y cols., 2023](#)), por lo que no es sorprendente que se centraran en la corrección de errores.

Creemos que en un trabajo futuro se debería investigar cuál es el comportamiento esperado de un corrector para su uso en el contexto en el que estamos trabajando. Es posible que para corregir textos escritos por estudiantes de nivel inicial no sea importante corregir todos los errores posibles y generar un texto natural sino restringirse a los errores que es razonable corregir para el nivel del estudiante.

Definir esto es importante para determinar cómo utilizar los corpus en el futuro y cómo se deben evaluar los sistemas para corregir textos.

5.3.5. Creación de una aplicación final

Durante el transcurso de este proyecto se discutió la posibilidad de crear una aplicación que pudiera ser utilizada por docentes en apoyo al proceso de corrección de textos. Si bien esto no pasó de ser una idea, creemos que la existencia de una aplicación como esta sería útil en dos sentidos:

- Corregir un texto utilizando un [LLM](#) no siempre genera el mismo resultado. Esto se debe a un elemento de aleatoriedad en la generación de textos. Es decir que distintas ejecuciones pueden dar resultados mejores o peores. Si los docentes pueden ejecutar en tiempo real al corrector, tendrían la posibilidad de ver distintas correcciones hasta encontrar una satisfactoria.
- Las correcciones generadas por la aplicación y luego revisadas por docentes podrían recopilarse y utilizarse como un corpus para re-entrenar y mejorar el propio sistema.

Tanto el campo de [GED](#) y [GEC](#) como el de los [LLMs](#) están en pleno desarrollo. Hay muchos caminos posibles tanto para mejorar la solución que presentamos como para implementar otros sistemas que aprovechen mejor las capacidades de los [LLMs](#), sin restringirse a la corrección de errores individuales. Consideramos que sería valioso continuar investigando en estas áreas.

Siglas

BEA Building Educational Applications. [6](#), [9](#), [11](#), [13](#)

BERT Bidirectional Encoder Representations from Transformers. [11](#), [37](#)

CoNLL Computational Natural Language Learning. [6–9](#), [13](#), [23](#)

GEC Corrección de Errores Gramaticales. [5](#), [9](#), [42](#), [53](#), [54](#), [56](#), [58](#)

GED Detección de Errores Gramaticales. [5](#), [53](#), [54](#), [56](#), [58](#)

LLM Gran modelo de lenguaje. [v](#), [vii](#), [2](#), [5](#), [11](#), [12](#), [18](#), [19](#), [21](#), [26–28](#), [31](#), [33](#), [34](#), [39](#), [41](#), [47](#), [49](#), [53–58](#)

PLN Procesamiento de Lenguaje Natural. [1](#), [2](#), [5](#), [10](#)

POS Part of Speech (Categoría Gramatical). [37](#), [38](#)

WER Word Error Rate. [viii](#), [24](#), [32](#), [33](#), [36](#), [38](#)

Glosario

accuracy El porcentaje de predicciones correctas hechas por un modelo. [17](#), [39](#)

embedding Representación vectorial de una texto. [18–20](#), [37](#), [48](#), [56](#)

gold standard Conjunto de datos de referencia contra los cuales se compara la respuesta de un modelo. [8](#), [41–43](#), [49](#), [52](#)

precision Porcentaje de predicciones positivas correctas hechas por un modelo. [17](#), [42](#), [46](#), [47](#), [49](#), [51](#), [52](#)

prompt Instrucción o texto inicial proporcionado a una herramienta generativa de IA para dirigir la generación de respuestas. [VII](#), [VIII](#), [2](#), [13](#), [18](#), [19](#), [26](#), [27](#), [32](#), [41](#), [47](#), [49](#), [54](#), [67–71](#)

recall Porcentaje de casos positivos que fueron capturados por un modelo. [17](#), [22](#), [42](#), [46](#), [47](#), [49](#), [51](#), [52](#)

token Unidad mínima de procesamiento de lenguaje. [18–20](#), [24](#), [33–38](#)

transformer Modelo de redes neuronales utilizado para el procesamiento de lenguaje natural. [2](#), [9](#), [19](#)

Referencias

- Bojar, O., Federmann, C., Fishel, M., Graham, Y., Haddow, B., Huck, M., . . . Monz, C. (2018, octubre). Findings of the 2018 conference on machine translation (WMT18). En O. Bojar y cols. (Eds.), *Proceedings of the third conference on machine translation: Shared task papers* (pp. 272–303). Belgium, Brussels: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W18-6401> doi: 10.18653/v1/W18-6401
- Brown, R., Paez, S., Herrera, G., Chiruzzo, L., y Rosá, A. (2023, mayo). Experiments on automatic error detection and correction for uruguayan learners of English. En D. Alfter, E. Volodina, T. François, A. Jönsson, y E. Rennes (Eds.), *Proceedings of the 12th workshop on nlp for computer assisted language learning* (pp. 45–52). Tórshavn, Faroe Islands: LiU Electronic Press. Descargado de <https://aclanthology.org/2023.nlp4call-1.6>
- Brown, R., y Páez, S. (2021). *Construcción de herramientas de soporte para corrección en enseñanza de inglés*. (FING)
- Bryant, C., Felice, M., Andersen, Ø. E., y Briscoe, T. (2019, agosto). The BEA-2019 shared task on grammatical error correction. En H. Yannakoudakis, E. Kochmar, C. Leacock, N. Madnani, I. Pilán, y T. Zesch (Eds.), *Proceedings of the fourteenth workshop on innovative use of nlp for building educational applications* (pp. 52–75). Florence, Italy: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W19-4406> doi: 10.18653/v1/W19-4406
- Bryant, C., y Ng, H. T. (2015, julio). How far are we from fully automatic high quality grammatical error correction? En C. Zong y M. Strube (Eds.), *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)* (pp. 697–707). Beijing, China: Association for Computational Linguistics. Descargado de <https://aclanthology.org/P15-1068> doi: 10.3115/v1/P15-1068
- Dahlmeier, D., Ng, H. T., y Wu, S. M. (2013, junio). Building a large annotated corpus of learner English: The NUS corpus of learner English. En J. Tetreault, J. Burstein, y C. Leacock (Eds.), *Proceedings of the eighth workshop on innovative use of NLP for building educational applications* (pp. 22–31). Atlanta, Georgia: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W13-1703>

- Felice, M., y Briscoe, T. (2015, mayo-junio). Towards a standard evaluation method for grammatical error detection and correction. En R. Mihalcea, J. Chai, y A. Sarkar (Eds.), *Proceedings of the 2015 conference of the north American chapter of the association for computational linguistics: Human language technologies* (pp. 578–587). Denver, Colorado: Association for Computational Linguistics. Descargado de <https://aclanthology.org/N15-1060> doi: 10.3115/v1/N15-1060
- Felice, M., Yuan, Z., Andersen, Ø. E., Yannakoudakis, H., y Kochmar, E. (2014, junio). Grammatical error correction using hybrid systems and type filtering. En H. T. Ng, S. M. Wu, T. Briscoe, C. Hadiwinoto, R. H. Susanto, y C. Bryant (Eds.), *Proceedings of the eighteenth conference on computational natural language learning: Shared task* (pp. 15–24). Baltimore, Maryland: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W14-1702> doi: 10.3115/v1/W14-1702
- Feuerriegel, S., Hartmann, J., Janiesch, C., y Zschech, P. (2024, 01 de Feb). Generative ai. *Business & Information Systems Engineering*, 66(1), 111-126. Descargado de <https://doi.org/10.1007/s12599-023-00834-7> doi: 10.1007/s12599-023-00834-7
- Granger, S. (1998). The computer learner corpus: A versatile new source of data for SLA research. In Granger, S. (ed.) *Learner English on Computer. Addison Wesley Longman : London New York*, 3-18.
- Grundkiewicz, R., Junczys-Dowmunt, M., y Heafield, K. (2019, agosto). Neural grammatical error correction systems with unsupervised pre-training on synthetic data. En H. Yannakoudakis, E. Kochmar, C. Leacock, N. Madnani, I. Pilán, y T. Zesch (Eds.), *Proceedings of the fourteenth workshop on innovative use of nlp for building educational applications* (pp. 252–263). Florence, Italy: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W19-4427> doi: 10.18653/v1/W19-4427
- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., . . . Sayed, W. E. (2023). *Mistral 7b*. Descargado de <https://arxiv.org/abs/2310.06825>
- Jurafsky, D., y Martin, J. H. (2024). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition with language models* (3rd ed.). Descargado de <https://web.stanford.edu/~jurafsky/slp3/> (Online manuscript released August 20, 2024)
- Mitchell, T. M. (1997). *Machine learning* (Vol. 1) (n.º 9). McGraw-hill New York.
- Mizumoto, T., Komachi, M., Nagata, M., y Matsumoto, Y. (2011, noviembre). Mining revision log of language learning SNS for automated Japanese error correction of second language learners. En H. Wang y D. Yarowsky (Eds.), *Proceedings of 5th international joint conference on natural language processing* (pp. 147–155). Chiang Mai, Thailand: Asian Federation of Natural Language Processing. Descargado de <https://aclanthology.org/I11-1017>
- Mohri, M., Rostamizadeh, A., y Talwalkar, A. (2012). *Foundations of machine*

- learning*. MIT Press. Descargado de <https://books.google.com/books?id=maz6AQAAQBAJ>
- Nesmachnow, S., y Iturriaga, S. (2019). Cluster-uy: Collaborative scientific high performance computing in uruguay. *Supercomputing. ISUM 2019. Communications in Computer and Information Science, vol 1151*.
- Ng, H. T., Wu, S. M., Briscoe, T., Hadiwinoto, C., Susanto, R. H., y Bryant, C. (2014, junio). The CoNLL-2014 shared task on grammatical error correction. En H. T. Ng, S. M. Wu, T. Briscoe, C. Hadiwinoto, R. H. Susanto, y C. Bryant (Eds.), *Proceedings of the eighteenth conference on computational natural language learning: Shared task* (pp. 1–14). Baltimore, Maryland: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W14-1701> doi: 10.3115/v1/W14-1701
- Ng, H. T., Wu, S. M., Wu, Y., Hadiwinoto, C., y Tetreault, J. (2013, agosto). The CoNLL-2013 shared task on grammatical error correction. En H. T. Ng, J. Tetreault, S. M. Wu, Y. Wu, y C. Hadiwinoto (Eds.), *Proceedings of the seventeenth conference on computational natural language learning: Shared task* (pp. 1–12). Sofia, Bulgaria: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W13-3601>
- Rozovskaya, A., Chang, K.-W., Sammons, M., y Roth, D. (2013, agosto). The University of Illinois system in the CoNLL-2013 shared task. En H. T. Ng, J. Tetreault, S. M. Wu, Y. Wu, y C. Hadiwinoto (Eds.), *Proceedings of the seventeenth conference on computational natural language learning: Shared task* (pp. 13–19). Sofia, Bulgaria: Association for Computational Linguistics. Descargado de <https://aclanthology.org/W13-3602>
- Sakaguchi, K., Napoles, C., Post, M., y Tetreault, J. (2016). Reassessing the goals of grammatical error correction: Fluency instead of grammaticality. *Transactions of the Association for Computational Linguistics*, 4, 169–182. Descargado de <https://aclanthology.org/Q16-1013> doi: 10.1162/tacl.a.00091
- Swanson, B., y Yamangil, E. (2012, junio). Correction detection and error type selection as an ESL educational aid. En E. Fosler-Lussier, E. Riloff, y S. Bangalore (Eds.), *Proceedings of the 2012 conference of the north American chapter of the association for computational linguistics: Human language technologies* (pp. 357–361). Montréal, Canada: Association for Computational Linguistics. Descargado de <https://aclanthology.org/N12-1037>
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... Scialom, T. (2023). *Llama 2: Open foundation and fine-tuned chat models*. Descargado de <https://arxiv.org/abs/2307.09288>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. En I. Guyon y cols. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Descargado de https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
- Volodina, E., Bryant, C., Caines, A., De Clercq, O., Frey, J.-C., Ershova,

- E., ... Vinogradova, O. (2023, mayo). MultiGED-2023 shared task at NLP4CALL: Multilingual grammatical error detection. En D. Alfter, E. Volodina, T. François, A. Jönsson, y E. Rennes (Eds.), *Proceedings of the 12th workshop on nlp for computer assisted language learning* (pp. 1–16). Tórshavn, Faroe Islands: LiU Electronic Press. Descargado de <https://aclanthology.org/2023.nlp4call-1.1>
- Xue, H., y Hwa, R. (2014, junio). Improved correction detection in revised ESL sentences. En K. Toutanova y H. Wu (Eds.), *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 2: Short papers)* (pp. 599–604). Baltimore, Maryland: Association for Computational Linguistics. Descargado de <https://aclanthology.org/P14-2098> doi: 10.3115/v1/P14-2098
- Yannakoudakis, H., Briscoe, T., y Medlock, B. (2011, junio). A new dataset and method for automatically grading ESOL texts. En D. Lin, Y. Matsumoto, y R. Mihalcea (Eds.), *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies* (pp. 180–189). Portland, Oregon, USA: Association for Computational Linguistics. Descargado de <https://aclanthology.org/P11-1019>
- Östling, R., Gillholm, K., Kurfah, M., Mattson, M., y Wirén, M. (2023). *Evaluation of really good grammatical error correction*. Descargado de <https://arxiv.org/abs/2308.08982>

Anexo A

Anexo 1

A.1. Algoritmo de Distancia de Levenshtein

```
int LevenshteinDistance(char str1[1..lenStr1], char str2[1..lenStr2])
    int i, j, cost

    for i from 0 to lenStr1
        d[i, 0] := i
    for j from 0 to lenStr2
        d[0, j] := j

    for i from 1 to lenStr1
        for j from 1 to lenStr2
            if str1[i-1] = str2[j-1] then cost := 0
            else cost := 1

            d[i, j] := minimum(
                d[i-1, j] + 1,    // deletion
                d[i, j-1] + 1,    // insertion
                d[i-1, j-1] + cost // substitution
            )

    return d[lenStr1, lenStr2]
```

A.2. Prompts

A.2.1. Prompt 1

I'm going to give you texts written by an ESL student whose mother tongue is Spanish. Mark and correct the errors in the texts.

There are different types of errors:

When the spelling is wrong, the error is Spelling.
 Mark these errors with <E_S> and the ending with </E_S>.
 For example, "<E_S>Thero</E_S> are three dogs."
 When the capital letter at the beginning of a sentence is used,
 the error is Beginning of Sentence.
 Mark these errors with <E_BOS> and </E_BOS>.
 For example, "I like dogs. <E_BOS>she</E_BOS> likes cats."
 When the pronoun "I" is written in lowercase, the error is Pronoun I.
 Mark these errors with <E_PI> and </E_PI>.
 For example, "<E_PI>i</E_PI> like dogs."
 When a proper name is written without capitalization, the error is Proper Noun.
 Mark these errors with <E_PN> and </E_PN>.
 For example, "I live in <E_PN>montevideo</E_PN>".
 When an article is missing, the error is Missing Article.
 Mark these errors with <E_MA> and </E_MA>.
 For example, "She eats <E_MA>apple</E_MA>".
 When there is an unnecessary article like "the" or "a",
 the error is Unnecessary Article.
 Mark the noun that has the unnecessary article with <E_UA> and </E_UA>.
 For example, "She like a <E_UA>books</E_UA>".
 When an incorrect verb form is used, the error is Verb Form.
 Mark the wrong verb form with <E_VF> and </E_VF>.
 For example, "She likes <E_VF>study</E_VF>".
 When the error is agreement between subject and verb,
 the error is Subject-Verb Agreement.
 Mark these errors with <E_SVA> and </E_SVA>.
 For example, "He <E_SVA>eat</E_SVA> apples."

Here is an example how it should work: Original phrase: "i like dog. she like cat. She likes study. She likes piza. She has got cat. She also likes a dogs. She is andrea."

The correction: "<E_PI>i</E_PI> like dog.
 <E_BOS>she</E_BOS> <E_SVA> like </E_SVA> cat.
 She likes <E_VF>study</E_VF>. She likes <E_S>piza</E_S>.
 She has got <E_MA>cat</E_MA>.
 She also likes a <E_UA>dogs</E_UA>. She is <E_PN>andrea</E_PN>."

Mark the errors in the original sentence when there's a difference between the original sentence and the corrected sentence.

This is the first text: "She is Andrea. She has 14 years old. She has got a yellow hair. She doesn't like sing. She like reading and pizza".

Mark the sentence with the errors you find and list the errors.

A.2.2. Prompt 2

I'm going to give you texts written by an ESL student whose mother tongue is Spanish. List all the errors you find and what type of error they are and then

give me the sentence with the errors marked with the <error>and </error>tag.

This is the first text, mark all the errors you find: "She is Andrea. She has 14 years old. She has got a yellow hair. She doesn't like sing. She like reading and pizza"

Original phrase: "i like dogs. she like cats. She likes study. She likes piza. She has got cat. She also likes a dogs. She is andrea."
The correction: "I like dogs. She likes cats. She likes studying. She likes pizza. She has got a cat. She also like dogs. She is Andrea."

The correction with error tags: "<error>i</error> like dogs. <error>she</error> <error> like </error> cats. She likes <error>study</error>. She likes <error>piza</error>. She has got <error>cat</error>. She also likes a <error>dogs</error>. She is <error>andrea</error>."

Please mark the error tag as in the previous example in this original phrase: "She is Andrea. She has 14 years old. She has got a yellow hair. She doesn't like sing. She like reading and pizza"

A.2.3. Prompt 3

I'm going to give you texts written by an ESL student whose mother tongue is Spanish to correct.

There are different types of errors:

1. <E_S>: Spelling Error.
For example, "<E_S>Thero</E_S> are three dogs."
2. <E_BOS>: Beginning of Sentence Error(when sentences start with lowercase).
For example, "<E_BOS>she</E_BOS> likes cats."
3. <E_PN>: Pronoun Error(when pronouns are in lowercase).
For example, "I live in <E_PN>montevideo</E_PN>".
4. <E_MA>: Article Mistake.
For example, "She eats <E_MA>apple</E_MA>".
- 5.<E_UA>: Unnecessary Article.
For example, "She like a <E_UA>books</E_UA>."
- 6.<E_VF>: Verb Form Error.
For example, "She likes <E_VF>study</E_VF>".
- 7.<E_SVA>: Subject-Verb Agreement Error.
For example, "He <E_SVA>eat</E_SVA> apples."

Here is an example how it should work: Original phrase: "she like cat. She likes study. She likes piza. She has got cat. She also likes a dogs. She is andrea."

The correction: "<E_BOS>she</E_BOS> <E_SVA> like </E_SVA> cat. She likes <E_VF>study</E_VF>. She likes <E_S>piza</E_S>."

She has got <E_MA>cat</E_MA>. She also likes a <E_UA>dogs</E_UA>. She is <E_PN>andrea</E_PN>."

This is the first text: "She is Andrea. She has 14 years old. She has got a yellow hair. She doesn't like sing. She like reading and pizza". Mark the sentence with the errors in the original phrase you find and list the errors.

A.2.4. Prompt 4

I'm going to give you texts written by an ESL student whose mother tongue is Spanish to correct.

There are different types of errors:

1. <E_S>: Spelling Error.
For example, "<E_S>Thero</E_S> are three dogs."
2. <E_BOS>: Beginning of Sentence Error(when sentences start with lowercase).
For example, "<E_BOS>she</E_BOS> likes cats."
3. <E_PN>: Pronoun Error(when pronouns are in lowercase).
For example, "I live in <E_PN>montevideo</E_PN>".
4. <E_MA>: Article Mistake.
For example, "She eats <E_MA>apple</E_MA>".
- 5.<E_UA>: Unnecessary Article.
For example, "She like a <E_UA>books</E_UA>."
- 6.<E_VF>: Verb Form Error.
For example, "She likes <E_VF>study</E_VF>".
- 7.<E_SVA>: Subject-Verb Agreement Error.
For example, "He <E_SVA>eat</E_SVA> apples."

This is the first text: "She is Andrea. She has 14 years old. She has got a yellow hair. She doesn't like sing. She like reading and pizza". Mark the errors in the original phrase you find and list the errors.

A.2.5. Prompt Version Final

This is a text written by an ESL student whose mother tongue is Spanish.

There are different types of errors:

1. <E_S>: Spelling Error.
2. <E_BOS>: Beginning of Sentence Error(when sentences start with lowercase).
3. <E_PN>: Pronoun Error(when pronouns are in lowercase).
4. <E_MA>: Article Mistake.
- 5.<E_UA>: Unnecessary Article.
- 6.<E_VF>: Verb Form Error.
- 7.<E_SVA>: Subject-Verb Agreement Error.

This is the first text: "she birhtday 14 years old i like pizza,right and ried and dont like,single,apple and fishing."

Give me the correct text first and then the list of errors.

A.2.6. Prompt usado para Mistral

original: <start>She is 14 years old. She is long and legs tail. She is pink t-shirt , whit shorts and whit trainers. She like read , pizza and maht.<end>
 corrected: <start>She is 14 years old. She is tall and has long legs. She is wearing a pink t-shirt, white shorts and white trainers. She likes reading, pizza and maths.<end>

original: <start>She has got a bike and a dog. She is wearing t-shirt, shorts and trainers. She can reading but she can't sing. She like eating pizza but she don't like apples.<end>
 corrected: <start>She has got a bike and a dog. She is wearing a t-shirt, shorts and trainers. She can read but she can't sing. She likes eating pizza but she doesn't like apples.<end>

original: <start>she is andrea. she is 14 years old. she jas wellow jeals. she like pizza. she nou to sing. she he does not like the apple.<end>
 corrected: <start>She is Andrea. She is 14 years old. She has yellow jeans. She likes pizza. She knows how to sing. She does not like apples.<end>

original: <start>TEXTO_A_CORREGIR<end>
 corrected: <start>

A.3. Textos usados para entrenar al modelo

id	Puntaje	Original	Corrección
12	3	She is 14 years old. She is long and legs tail. She is pink t-shirt , whit shorts and whit trainers. She like read , pizza and maht.	She is 14 years old. She is tall and has long legs. She is wearing a pink t-shirt, white shorts and white trainers. She likes reading, pizza and maths.
43	4	She has got a bike and a dog. She is wearing t-shirt, shorts and trainers. She can reading but she can't sing. She like eating pizza but she don't like apples.	She has got a bike and a dog. She is wearing a t-shirt, shorts and trainers. She can read but she can't sing. She likes eating pizza but she doesn't like apples.
144	3	she is andrea. she is 14 years old. she jas wellow jeals. she like pizza. she nou to sing. she he does not like the apple.	She is Andrea. She is 14 years old. She has yellow jeans. She likes pizza. She knows how to sing. She does not like apples.

Tabla A.1: Textos usados para entrenar el modelo