TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Desarrollo de sistema de resolución de rompecabezas en ASP mediante el uso de LLM

Estudiante: Pedro Pazos Curra

Dirección: José Pedro Cabalar Fernández

A Coruña, agosto de 2024.

Resumen

Los Large Language Model (LLM) han recibido una gran popularidad en los últimos años por su capacidad de generar texto de apariencia orgánica. Sin embargo, una de sus restricciones más flagrantes es su incapacidad para realizar inferencias complejas o resolver determinadas cuestiones necesitadas de un razonamiento profundo. Para tratar este problema, se propone combinar un LLM con Answer Set Programming (ASP), un formalismo de programación lógica usado para la resolución declarativa de problemas. Se implementa un sistema neurosimbólico que combina LLM con ASP en el contexto de la resolución de determinados tipos de rompecabezas. Se consigue, además, incorporar un sistema con interfaz Web para mostrar a un potencial usuario una salida gráfica y una descripción en texto en Lenguaje Natural (LN) de las soluciones halladas.

Abstract

Large Language Model (LLM)s have become very popular in recent years because of their ability to generate organic-looking text. However, one of their most blatant restrictions lies in their inability to perform complex inferences or to solve certain questions that require deep reasoning. To address this issue, it is proposed to combine an LLM with Answer Set Programming (ASP), a logic programming formalism used for declarative problem solving. A neurosymbolic system is implemented combining LLM with ASP in the context of solving certain types of puzzles. As an adittional feat, a system with a Web interface has been incorporated to show a potential user both a graphical output and a natural language text description of the solutions found.

Palabras clave:

- Grandes Modelos de Lenguaje (LLM)
- Programación de Conjuntos de Respuestas (ASP)
- Procesamiento de Lenguaje Natural (NLP)
- Resolución de Problemas
- Representación de Conocimiento

Keywords:

- Large Language Models (LLM)
- Answer Set Programming (ASP)
- Natural Language Processing (NLP)
- Problem-solving
- Knowledge Representation

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

Los lenguajes orientados al razonamiento lógico comportan un interés superlativo por su eficacia y su potencia. A diferencia de otros paradigmas de programación, poseen una metodología alternativa en cuanto a la forma de plantear un problema a través de su descripción en lugar de a través de su resolución algorítmica, lo cual es considerado muy atractivo como materia de estudio. En adición, los modelos de lenguaje extensos (LLM), se han granjeado una enorme popularidad en los últimos años, gracias a la irrupción en la sociedad de diversos modelos ahora universalmente conocidos, véase ChatGPT (OpenAI), Gemini (Google) o LLaMa (Meta). Son, en la actualidad un tema candente en la esfera académica por sus virtudes técnicas pero también por los desafíos educativos que su figura presenta[1]. Ambas tecnologías son tratadas con su debida profundidad en sendos apartados del presente documento(2.1, 2.2).

De acuerdo con la bibliografía al respecto (cuestión explayada en la Sección 2.2), la falta de razonamiento profundo o *System 2 thinking* se encuentra entre las limitaciones actuales de los LLM. Siguiendo esta línea, el equipo localiza el potencial de realizar un sistema que ocupe la labor no plenamente satisfecha en aquellas inferencias más desafiantes; mejorando la eficacia de este tipo de modelos sin alterar su funcionamiento de cara al usuario.

1.1 Objetivos

Como cometido principal, el presente Trabajo de Fin de Grado (TFG) tiene el abordaje del problema identificado a través del diseño y realización de un sistema con capacidad de resolver, representar y explicar la solución alcanzada de determinados rompecabezas mediante una aproximación neurosimbólica que haga uso conjunto de un Large Language Model (LLM) y de Answer Set Programming (ASP). Se busca, además, abstraer al usuario final de la incor-

poración del lenguaje intermedio, de modo que dicho cliente tuviese la impresión de plantear su cuestión como lo haría con cualquier otro de los grandes modelos de lenguaje del mercado. Otro de los objetivos es el de hacer de dicho sistema una solución fiable y eficaz en su dominio, esto es, con una proporción de fallos sensiblemente menor que un supuesto análogo sin la incorporación de ASP ni la implementación del propuesto sistema neurosimbólico. Como última intención, se persigue la implementación de un módulo gráfico tal que el usuario pueda comprobar visualmente la solución a su problema propuesto, además de poder solicitar una descripción textual de dicha solución.

No está abarcado en el alcance de la tarea el desarrollo de un modelo de lenguaje *ad hoc* como tampoco lo está la creación de un sistema de propósito resolutor de rompecabezas universal, si bien pudiesen éstos proponerse como candidatos a un potencial trabajo futuro.

Para lograr los objetivos mencionados, el equipo propone un sistema en cuatro fases dependientes pero no necesariamente secuenciales de extremo a extremo. En primer lugar (1), el modelo de lenguaje tiene la tarea de representar la entrada, en Lenguaje Natural (LN), al lenguaje ASP, en la forma de predicados lógicos. A continuación (2), e incorporando esta producción al código ASP, se tiene la capacidad de ejecutar el programa y resolver el puzzle en cuestión a través del *solver* ASP, pues contiene de forma estática e interna el conocimiento de las reglas del rompecabezas elegido. En tercer lugar (3), la salida del programa ASP, que contiene los modelos mínimos del programa lógico (esto es, el *answer set*), se transfiere al LLM para ser articulado nuevamente en forma de LN y enviar al usuario la explicación de la solución alcanzada. En último lugar (4), es empleado un módulo de imagen para representar de forma gráfica los estados inicial y final del rompecabezas. La respuesta otorgada al usuario, tanto en forma de texto (3) como gráfica (4), se plantean con un carácter opcional, dejando a elección del usuario indicar si desea que sea expuesta una de las resoluciones o ambas.

1.2 Organización del documento

El presente documento, siguiendo el estándar académico, comienza con el actual Capítulo introductorio 1, donde se contextualiza la situación que rodea un problema determinado, se declaran las pretensiones del proyecto propuesto y las soluciones ya existentes en el mercado para resolverlo. El segundo (2) Capítulo estudia los fundamentos, esto es, todo aquel conocimiento, técnicas o metodologías cuyo conocimiento previo se considera relevante para comprender lo realizado en el trabajo.

El bloque conformado por los Capítulos del tercero al sexto comprende el núcleo de la labor realizada. En orden: la planificación y presupuesto (Capítulo 3), la interfaz de usuario (Cap. 5), el servidor (Cap. 6) y, por último, el sistema neurosimbólico (Cap. 7). En el Cap. dedicado a la planificación y el presupuesto, se elaboran las estimaciones de la gestión del proyecto en términos económicos y temporales. En el capítulo sobre el *frontend*, se detalla lo referente al diseño y maquetación de lo que a ojos del usuario es capa exterior del sistema. En el Cap. posterior, dedicado a la capa del servidor, es expresado el desarrollo del aparato intermedio entre la interfaz y el sistema neurosimbólico. En el Capítulo 7, finalmente, se desglosa el desarrollo del sistema neurosimbólico, con sendas divisiones lógicas del proceso.

A continuación, en el Cap. 8, se llevan a cabo diversas evaluaciones sobre el rendimiento de la solución alcanzada: métricas del sistema; ventaja sobre la utilización rasa del LLM; soporte sobre navegadores o sistemas operativos; etcétera. En última instancia, para rememorar acerca del trabajo realizado, sopesar el potencial futuro y declarar las conclusiones finales, la memoria se cierra con el Cap. 9.

Capítulo 2

Antecedentes

En esta sección es abordada la tarea de presentar una base teórica de todos los elementos existentes que, si bien aportan valor al proyecto, no forman parte del trabajo realizado en el mismo. De la misma forma, se expone en la presente sección el marco teórico preciso para dotar al lector de la capacidad de comprender en su plenitud la labor posteriormente expuesta.

2.1 *Answer Set Programming* y *Clingo*

Answer Set Programming (ASP)[2][3] es un lenguaje lógico declarativo perteneciente a la familia de lenguajes de programación lógica. En él, se han de establecer un conjunto de construcciones, llamadas reglas, de la forma $a_1 \mid ... \mid a_n \leftarrow b_1, \ ... \ , \ b_m$. En el lado izquierdo de la misma está el consecuente, donde se encuentran los átomos $a_i \ (0 \leq i \leq n)$. El lado derecho, el antecedente, alberga los literales $b_i \ (0 \leq i \leq m)$. Los literales pueden ser verdaderos (b_i) o falsos (*not* b_i). Si una regla carece de la parte izquierda, se dice que es una restricción. Si no goza, en cambio, de su parte derecha, cobra el nombre de hecho; pues, al ser una aseveración incondicional, es axiomática en el sistema.

ASP tiene como objetivo, mediante un subprograma interno de resolución independiente de contexto (*solver*), encontrar los conjuntos de configuraciones de literales (modelos clásicos, recurriendo al análogo en lógica clásica) tal que se cumplan todas las restricciones y condiciones impuestas entre los diferentes átomos. De entre estos modelos, el más pequeño es el llamado *answer set* o modelo estable, y un programa lógico arbitrario puede tener varios o ninguno. Un *answer set* es siempre un modelo del sistema. En otras palabras, un conjunto de átomos X conforman el *answer set* de un programa positivo $\prod$ sólo si $C_n((\prod)^x) = X$, siendo $C_n((\prod)^x)$ el modelo clásico más pequeño de $\prod$.

Dichos átomos y reglas son la base de un código escrito en ASP; que a través de la sintaxis ”antecedente :- consecuente” y determinados *macros* (*#count{...}* para contar el número de átomos, *#min{...}* y *#max{...}* para encontrar valores límite de los pertenecientes a un átomo, etcétera), permiten al desarrollador definir cualquier programa de lógica no monótona (esto es, aquella lógica cuyo número de modelos puede decrecer cuanta más información se añada a un problema).

2.1.1 Clingo

Clingo es uno de los proyectos de Potassco (*Potsdam Answer Set Solving Collection*), una agrupación formada en la Universität Potsdam cuyo objetivo es la elaboración de diversas herramientas basadas en el Answer Set Programming. Clingo es una herramienta que combina las herramientas de *grounding* (gringo) y *solving* (clasp) para programas escritos en ASP. El proceso de *grounding* es necesario para permitir el uso de variables de primer orden, de modo que el *grounder* pueda convertirlas a variables libres y el *solver* pueda hallar los modelos que cumplan con todas las reglas impuestas. En definitiva, Clingo es un sistema que permite cómodamente ejecutar un programa ASP. En la actualidad, es el software estándar para trabajar con ASP en el ámbito académico.

2.2 Gran Modelo de lenguaje

2.2.1 Definición

Un Large Language Model (LLM) es un tipo de modelo fundacional, que a su vez es una de las clases de los modelos de Inteligencia Artificial (IA). Son generalmente referidos como aquellos sistemas de Red de Neuronas Artificiales (RNA) adiestrados mediante aprendizaje sin supervisión sobre conjuntos de datos en forma de texto no etiquetados y de una dimensión considerable.

Hoy en día, la arquitectura más comúnmente empleada para entrenar estos modelos es la de los transformadores, que gozan de un buen rendimiento en los conjuntos de datos secuenciales (en este caso, *tokens*, que son representaciones numéricas comprimidas de una palabra o una sección de una). Los transformadores son un tipo de arquitectura (ilustrada en la Figura 2.1) perteneciente al denominado *Deep Learning (DL)*. Son presentados en 2017 por un equipo de ocho científicos de Google en el artículo *”Attention is All You Need”* [4]. Este método se basa en los sistemas de atención, que buscan emular el comportamiento de la atención en el propio cerebro humano. De este modo, y a grandes rasgos, la técnica aplicada determina dada una entrada previa (la ventana de contexto leída, en este caso) y una entrada actual (un

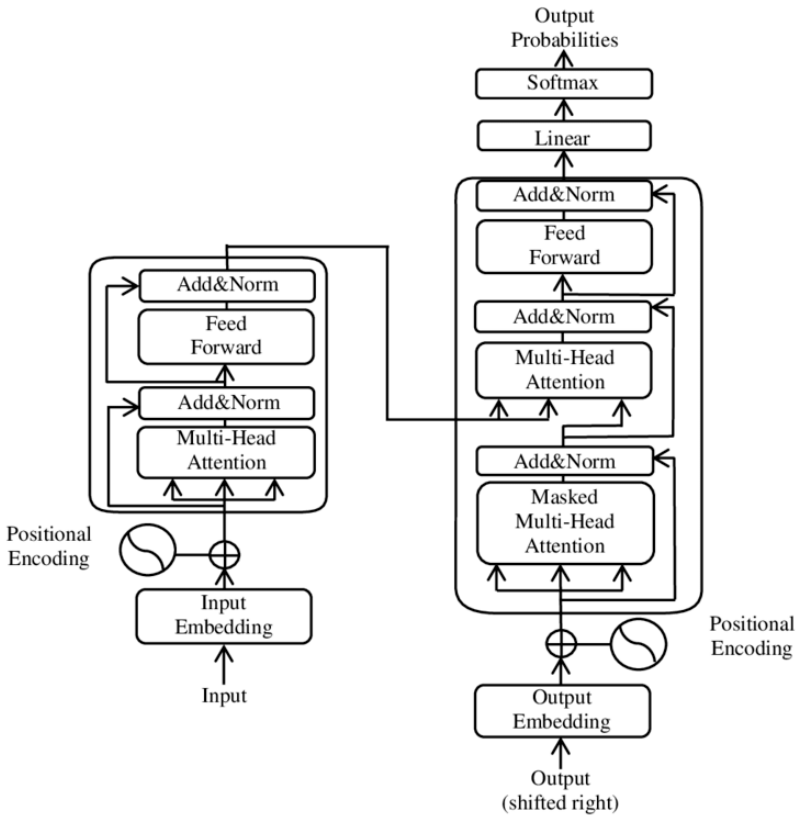

Figura 2.1: Arquitectura del transformador

token) un conjunto numérico compuesto por las probabilidades de aparición de diferentes tokens a continuación. Este flujo permite al sistema ”coser” una cadena de tokens, habiéndose estimado a cada una de las palabras una relevancia sensible con relación a sus predecesoras.

2.2.2 *K-Shot learning, In-context learning* y *Fine-tuning*

Ampliando los conocimientos básicos sobre los modelos extensos de lenguaje, cabe aclarar algunos términos que eventualmente cobrarán su debido protagonismo por su necesidad en la tarea de especializar en un ámbito un modelo de lenguaje previamente entrenado. Estos conceptos son, de base, diferentes técnicas del proceso común de seguir aportando datos a un modelo que ya ha terminado su fase de entrenamiento (de ahí el sobrenombre de los modelos *pre-entrenados*; que son los generalmente usados para este tipo de procedimientos).

El primer método, conocido como *k-shot learning*, consiste en otorgar en la propia entrada de un modelo pre-entrenado un número K de ejemplos etiquetados para, si procede, hacer-

lo más específico a los desafíos del trabajo a realizar. Entre sus variantes más comúnmente ubicables en la bibliografía se encuentran *zero-shot* (inferencia realizada sin ejemplos dados), *one-shot* (inferencia con un sólo ejemplo aportado) y *few-shot* (abstracción que indica la no pertenencia a los dos términos anteriores sin indicar el número de ejemplos). *k-shot learning* es un término general que puede ser referido como cualquier sistema inteligente basado en aprendizaje supervisado. Sin embargo, cuando el término se usa en el ámbito de los LLM, tiene un significado indistinto del *in-context learning* (comentado a continuación), con el matiz de indicar el número de ejemplos en el mismo término, con la forma "*5-shot learning*", por ejemplo.

Como segunda anotación, el propio *in-context learning* (aprendizaje en ventana de contexto). Es un *k-shot learning* aplicado en específico a LLM, en el que esta adición de ejemplos etiquetados se hace en el propio *prompt*, la entrada del usuario (comúnmente llamada ventana de contexto, de ahí el nombre). Brown et al.[5] aseveran en su experimento al respecto que estas técnicas mejoran de forma notable el rendimiento de los modelos en un amplio rango de áreas, desde la resolución de anagramas hasta los problemas aritméticos. Este procedimiento es también ampliamente conocido como *Prompt Engineering*; y, pese a ser una fórmula menos académica, es usada de forma equivalente en la jerga del tema.

El término más común en la bibliografía de entre los mencionados es el *fine-tuning*. El *fine-tuning*, al igual que el *in-context learning*, es una especie de post-entrenamiento con una nueva base de datos más específica para la futura aplicación del modelo, pero a diferencia del de éste, el entrenamiento del primero se realiza con una herramienta especializada y, lo que es más importante, la salida resulta en otro modelo diferente al inicial y con diferentes ajustes de sus parámetros internos. Puede realizarse o bien a todas las capas de neuronas o a sólo algunas de ellas, manteniendo los pesos de las otras estáticos; o "congelados", según el argot del campo. Es una técnica muy extendida tanto entre profesionales del oficio como entre académicos por su efectividad y buenos resultados en comparación con los otros métodos citados[6]. En cambio, y como asegura Matthew McMullen para Data Science Central[7], es severamente intensiva en recursos, al precisar iteraciones adicionales del algoritmo de entrenamiento del modelo; así como más compleja de realizar frente al aprendizaje dentro del contexto. A cambio, por norma general, se produce un modelo con un conocimiento más específico sobre los datos aportados y sus producciones se presupondrán más precisas sobre esa área concreta.

Actualmente, en el procedimiento se manejan principalmente dos métodos alternativos. En primer lugar, el Supervised Fine-Tuning (SFT)[8], consistente en la reducción iterativa de una función de pérdida correspondiente a los desaciertos del conjunto de entrenamiento. En

segundo lugar, más reciente, el *Reinforcement Learning Fine-Tuning* (Fine-Tuning mediante entrenamiento por refuerzo)[9], mediante el cual se pondera la bondad de un ejemplo en a través de una función determinista que asigna un valor de recompensa a una salida del sistema en base a cuánto se asemeja al valor esperado (premiado con la mayor retribución). La complejidad de esta última variante recae en la asignación de dicha función.

2.3 Métricas de Learning

Se considera conveniente aportar al lector una acotada base teórica en cuanto a algunas métricas que comúnmente son usadas en el campo del aprendizaje máquina, si bien son también expresiones empleadas en el amplio ámbito de la estadística. Estos valores cobrarán su debida importancia en el capítulo de evaluación (Capítulo 8)

2.3.1 Matriz de confusión

Una matriz de confusión es una matriz de N filas y N columnas (típicamente, N=2) empleada para clasificar las muestras de determinada medición con el objetivo de compararlas con su valor esperado de antemano. Las filas representan la clasificación fehaciente de las instancias aportadas mientras que las columnas son las clases predichas por el sistema o técnica a prueba. Queda, de este modo, una matriz semejante a la Tabla 2.1.

	Positivo dado	**Negativo dado**
Positivo predicho	Verdadero positivo (TP)	Falso positivo (FP)
Negativo predicho	Falso negativo (FN)	Verdadero negativo (TN)

Tabla 2.1: Matriz de confusión a modo de ejemplo con *alias* en las variables. Nótese: el ejemplo muestra una clasificación binaria, pero podría aumentar la cardinalidad de forma simétrica apareciendo más clases candidatas.

Una vez presentada la matriz de confusión, resulta de interés anticipar algunas de las métricas inferibles en base a ella que harán su debida aparición en su correspondiente Capítulo 8.

2.3.2 *Precision*

La precisión (*precision* inglesa; nótese la diferencia con *accuracy*, homógrafa en la traducción literal al castellano), también conocida como valor predictivo positivo, representa de una

medición el número de verdaderos positivos frente a todos aquellas instancias clasificadas como positivo. En otras palabras, determina la agudeza de un sistema al decir que una instancia es positiva. El valor predictivo positivo tiene un papel clave en aquellos sistemas donde resulta crucial poder asegurar que un positivo efectivamente lo es, sufriendo especialmente los casos de falsos positivos. Un ejemplo de un sistema con estos parámetros puede ser una biopsia de cáncer de mama.

La *precision* se calcula de la siguiente manera: $\frac{TP}{TP+FP}$

2.3.3 *Recall*

El *recall*, también conocido como sensibilidad o probabilidad de detección, mide cuántos de los elementos positivos son efectivamente etiquetados como uno. Esto es, puede determinar la probabilidad que tiene un sistema de no ignorar un caso positivo cuando se le presenta. Esta métrica tiene una gran relevancia en sistemas que sufren prominentemente ante los falsos negativos, como, por ejemplo, las pruebas del VIH.

El *recall* se calcula de la forma: $\frac{TP}{TP+FN}$

2.3.4 *F1-Score*

El *F1-Score* es la media armónica (nº observaciones / sumatorio del inverso de las observaciones) entre *precision* y *recall*. Sirve como una forma de representar ambos valores en una sóla métrica para facilitar el proceso de trabajar con ellas.

La *F1-Score* se calcula con la fórmula: $\frac{2TP}{2TP+FP+FN}$

2.4 *State of the Art*

En la actualidad, existen numerosos proyectos que estudian y ponderan el rendimiento de diferentes LLM trabajando en conjunto con ASP. En primer lugar, resultan de ineludible mención los realizados por el equipo de investigación del laboratorio Azreasoners. Este equipo de científicos forma parte del grupo de inteligencia artificial de la *School of Computing, Informatics and Decision Systems Engineering* de la *Arizona State University* (ASU), en Estados Unidos (EE.UU). Según su propia página web[10], sus proyectos siguen, en temática, la línea de la representación de conocimiento, la programación lógica, el razonamiento de sentido común, entre otros; materias en conveniente intersección con el presente proyecto.

De entre sus investigaciones, aquellas que se valoran como de un interés compulsivo son ”Coupling Large Language Models with Logic Programming for Robust and General Reasoning from Text” [11] y ”Leveraging Large Language Models to Generate Answer Set Programs” [12]. En ambas se aborda y discute el desempeño de las LLM para el contexto concreto de la adaptación al lenguaje de Answer Set Programming (ASP), con objetivos y metodologías sensiblemente diferentes entre una y otra. Comparten, no obstante, el planteamiento: los LLM son, por su naturaleza, sobresalientes en el llamado *System 1 thinking* (”pensamiento de sistema 1”: rápido, automático, subconsciente) y deficitarios en el *System 2 thinking* (”pensamiento de sistema 2”: lento, sosegado, consciente). En un intento por potenciar el razonamiento lógico, proponen la incorporación de ASP al circuito inferencial, buscando robustecer esa debilidad localizada. Esta incapacidad de los LLM de llevar a cabo razonamiento complejo, así como procesar dilemas éticos o problemas matemáticos es una observación ampliamente compartida en la comunidad científica, y abunda la bibliografía [13][14][15] tratando este sujeto.

En el primer trabajo de los dos mencionados de Azreasoners[11], se propone la adaptación de un LLM como parser semántico de LN a ASP mediante *in-context learning*. El proyecto tiene el objetivo de igualar o superar en diversas métricas de Procesamiento de Lenguaje Natural (NLP) a otros métodos específicos del *state of the art*. El enfoque principal del estudio es el interés de la aproximación neuro-simbólica del conjunto ASP-LLM.

En el segundo estudio referenciado[12] y a través del *fine-tuning* se busca mejorar el rendimiento de las LLM en materia de razonamiento lógico. Se aproxima mediante la incorporación de un resolutor ASP en medio del proceso que se encarga del aparato lógico, mientras que es el modelo el que cubre la parte del lenguaje natural. Su diferencia, sin embargo, con el escrito mencionado previamente [11], yace en el enfoque del presente, pues se le da prioridad en el estudio a la capacidad de los modelos de generar lenguaje ASP, dando solo un repaso al potencial de la aproximación neuro-simbólica. La experimentación se divide en tres partes: generación de predicados, extracción de constantes y generación de reglas. Este caso concreto resulta ser una excelente lectura como estudio previo al proyecto, debido a la total afinidad con el sistema documentado en la presente memoria. Se considera de valor la aportación que hace el escrito en cuanto a diferentes formatos de entrada, modelos y técnicas a aplicar para mejorar los resultados obtenidos. El formato homogéneo, por ejemplo, favorece las mediciones, así como parafrasear los ejemplos, véase, ”El gallo canta después del hombre, que no lo hace” es un predicado con menor adherencia para el sistema que un hipotético enunciado fraccionado ”El gallo canta”, ”El hombre no canta” y ”El gallo canta después del hombre”. En lo referido a los modelos, los responsables advierten una sensible ventaja inferencial del mo-

delo GPT4 frente a GPT3.

Adicionalmente (y cronológicamente posterior), Borroto et al.[16] lanzan un documento que estudia la composición automática de programas ASP partiendo de LN y Lenguaje Natural Controlado (CNL). Los CNL[17] son subconjuntos del lenguaje natural con un vocabulario restringido, especialmente indicados para usos computacionales. Para ello, el equipo desarrolla la herramienta NL2ASP y utiliza la previamente desarrollada CNL2ASP [18]. Como sus nombres insinúan, el primero tiene el cometido de traducir Lenguaje Natural a ASP mientras que el segundo lo hace desde Lenguaje Natural Controlado. Presentan, además, un *dataset* especializado en especificaciones de problemas basados en grafos, llamado NL2CNL, donde recogen 494 pares de traducción LN-CNL.

En febrero de 2023, Abhiramon Rajasekharan et al.[19], desde la universidad estadounidense de Texas, publican un documento muy similar en objetivos a los observados (intención de crear una variante de los LLM específica que confíe en ASP para el razonamiento complejo), con la peculiaridad de que este documento aborda de manera más profusa la explicabilidad y la fiabilidad de este sistema basado en LLM. Para este cometido, el equipo propone el marco ”STAR” (Semantic-parsing Transformer and ASP Reasoner). Al igual que en los ejemplos anteriores, se decantan por el *fine-tuning* frente al aprendizaje dentro del contexto. Un mes después, el mismo equipo lanza un agente conversacional[20] usando este marco de desarrollo STAR, con unos resultados, según los autores, todavía con cierto avance a la vista.

Como última anotación, un caso de una aplicación práctica en el campo de la robótica; en junio de 2023 Yan Ding et al.[21] presentan un escrito que abarca el desarrollo iterativo de una tecnología de planificación de tareas y movimiento. Esta es puesta en práctica sobre lo que podría ser uno de los robots que hoy en día un cliente pudiera encontrar en un local realizando sus labores de mesero. Bautizan a esta tecnología como LLM-GROP. Al igual que los casos anteriores, simbiotiza la capacidad de aprendizaje de sentido común de un LLM con el razonamiento complejo de un programa ASP. De acuerdo con el escrito presentado, el sistema implementado supera de forma holgada a otras tecnologías del mercado, con menor tiempo de ejecución y menor número de errores.

Como última anotación, cabe hacer referencia a una pequeña colección de aquellos trabajos académicos[22][23][24] que, si bien no son de una absoluta coincidencia temática, presentan una adyacencia notable como para que sus avances hayan tenido un papel relevante en el desarrollo del presente proyecto. No se cedió a ahondar en sus artículos con la debida profundidad, al no ser la actividad central de la tarea que se ocupa.

Capítulo 3

Planificación y presupuesto

Son evaluados en la presente división los costes económicos estimados de todos los recursos de los que se dispone en la elaboración del proyecto, tanto los humanos como los materiales (licencias de software, equipo informático empleado). Del mismo modo, se describe de forma aproximada la planificación temporal de las tareas a realizar a lo largo del proceso de desarrollo, pruebas, documentación, etcétera.

3.1 Presupuesto

En lo referente a la estimación del desembolso que el proyecto acarrea, son segregados los recursos por razón de tipo de recurso para finalizar el ejercicio con un sumatorio de la cantidad total. En primer lugar, se estudian en la tabla 3.1 los recursos humanos necesarios para llevar a término las labores consideradas y sus costes asociados. Del mismo modo, en la tabla 3.2 son indicados aquellos recursos materiales de obligada adquisición para la realización del trabajo y sus respectivos costes. En último lugar, la tabla 3.3 une ambas estimaciones para alcanzar la suma final, correspondiente al presupuesto total de la realización del proyecto.

Los costes unitarios asignados a los recursos humanos han sido elegidos siguiendo como orientación las Tablas de Retribución de Personal de Administración y Servicios de la *Universidade Da Coruña*.

3.1.1 Recursos humanos

Mano de obra	Tiempo dedicado	Coste unitario (€/h)	Total (€)

Desarrollador	734,5h	18,00	13.221,00
Analista 1	40h	23,00	920,00
Analista 2 (Tutor)	12h	23,00	276,00
Usuarios de prueba	1,5 h	15,00	22,50
Total	-	-	14.439,50

Tabla 3.1: Desglose de coste estimado de recursos humanos

3.1.2 Recursos materiales

Concepto	Cantidad	Coste unitario (€)	Total (€)
Ordenador de sobremesa	1 ud.	2.873,89	2.873,89
Ordenador portátil	1 ud.	384,99	384,99
Licencia Windows 10 Home	2 ud.	145,00	290,00
Total	-	-	3553,88

Tabla 3.2: Desglose de coste estimado de recursos materiales

3.1.3 Presupuesto total

Concepto	Coste (€)
Recursos humanos	14.439,50
Recursos materiales	3.553,88
Total	17.993,38

Tabla 3.3: Estimación total de coste del proyecto

3.2 Planificación

Establecido un presupuesto estimado, resta para el análisis preliminar la planificación aproximada en clave temporal de las tareas a realizar, cometido desglosado en la tabla 3.4.

Tarea	Horas / día	Días invertidos	Horas totales
Análisis	2h	20d	40h
Desarrollo Frontend	6h	17d	102h
Des. Backend	5h	5d	25h
Des. Sistema Neurosim.	6,5h	45d	292,5h
Evaluación y pruebas	3.5h	20d	70h
Documentación / memoria	7h	35d	245h
Total	-	-	774,5h

Tabla 3.4: Planificación del proyecto

Capítulo 4

Estructura del sistema

Previo comienzo del desarrollo del proyecto el equipo se propone la tarea de diseñar y plantear el sistema y arquitectura a usar. En primera instancia, son formulados los casos de uso que el sistema debe implementar, con la correspondiente elaboración del diagrama de casos de uso (presentado en la Figura 4.1). En su uso habitual, se espera que el usuario realice una consulta al sistema y reciba la solución a su problema con una descripción textual de la misma y/o con una representación gráfica, por lo que es evidente que una línea de casos de uso debe ser destinada a cumplir esta labor. De este modo se generan los casos de uso ”Realizar consulta al sistema”, extendida a ”Realizar consulta gráfica” y ”Realizar consulta texto”. Además, debido a la intención de que las reglas de los diferentes puzzles implementados estén integradas en el sistema de forma estática, el equipo estima que el usuario debería elegir en la propia interfaz qué puzzle desea resolver, con su consecuente caso de uso ”Elegir Puzzle”. Todas estas tareas básicas del sistema pueden ser realizadas por un usuario sin ningún tipo de formación específica ni conocimientos informáticos, por lo que se enlazan al actor ”Usuario”, siendo éste un cliente raso del sistema.

De forma adicional, se plantean diversas funcionalidades extra que un usuario con determinados conocimientos de informática o ASP pudiera llevar a cabo. Se genera, de este modo, el actor ”Experto ASP”. Dicho actor pudiera realizar acciones más abstractas en el sistema. Primeramente, bajo la suposición de que las imágenes usadas en el módulo gráfico son incorporadas de forma estática al sistema (como se comenta en su propia sección del Capítulo 7), podría este usuario experto añadir o eliminar imágenes para modificar sus salidas gráficas. Además, por poseer conocimientos en ASP, tendría la capacidad de alterar bajo su diligencia las reglas de los puzzles ya implementados. No podría, sin embargo, añadir ni eliminar puzzles, pues se valora que los mismos deben estar engranados en el resto del sistema, dicho de cierto modo. Por último, y siguiendo la misma lógica que en el caso de las imágenes, este usuario podría desear modificar el contexto dotado al LLM; cambiando las órdenes dadas al

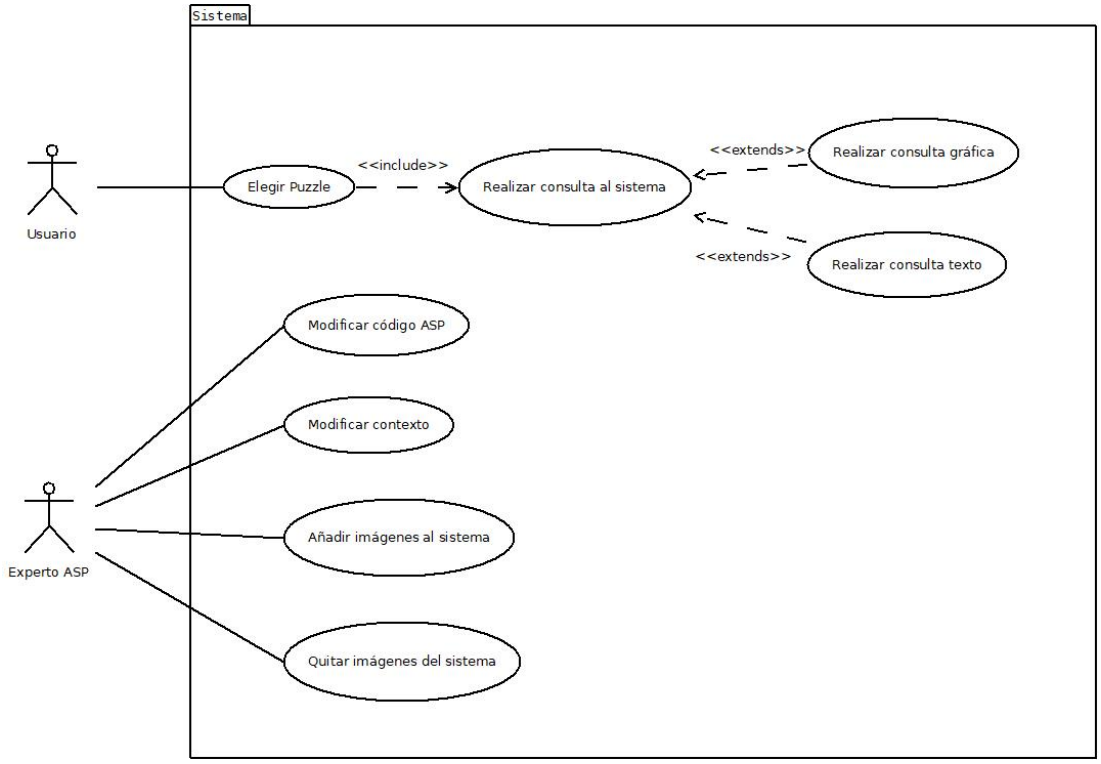

Figura 4.1: Diagrama de casos de uso.

mismo, por si buscase optimizar los resultados en su interés.

Recapitulando, este nuevo actor ”Experto ASP” tendría las capacidades de llevar a cabo los casos de uso ”Modificar código ASP”, ”Modificar contexto”, ”Añadir imágenes al sistema” y ”Quitar imágenes del sistema”.

A continuación, el equipo analiza la arquitectura propuesta para el sistema en su conjunto. Se planifican de este modo tres componentes principales: la ”Interfaz Web”, el ”Servidor”, y el ”Sistema Neurosimbólico”; el componente auxiliar ”Base de Datos”, entendiéndose como el repositorio con todos los recursos usados (imágenes, contexto, etcétera); y los componentes externos al sistema ”API LLM”, ”*Solver* ASP” y ”Paquete de manipulación de imágenes”, cuyas funcionalidades son comentadas en sus respectivas secciones del capítulo del sistema neurosimbólico (Capítulo 7).

El equipo estima como una buena aproximación emplear una arquitectura en capas para el desarrollo del sistema completo, si bien puede diferir de la arquitectura interna del sistema neurosimbólico (comentada en la Sección 7.1). Las capas de la arquitectura del sistema ge-

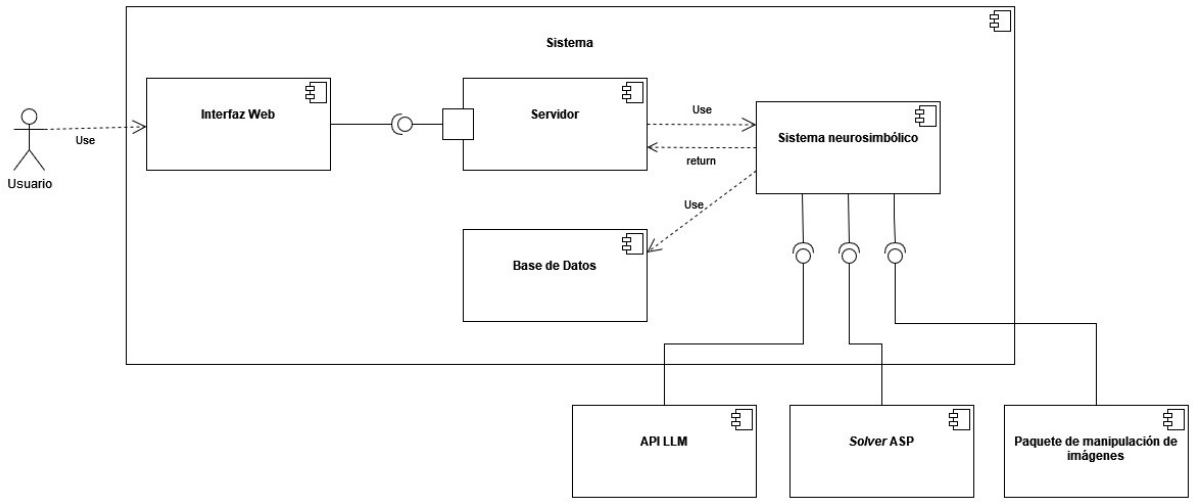

Figura 4.2: Diagrama de componentes de alto nivel.

neral serían los tres componentes principales previamente enumerados, donde cada petición al sistema lo atravesaría por completo ida y vuelta; yendo desde la Interfaz Web al Sistema Neurosimbólico pasando por el Servidor, y volviendo a la Interfaz Web de nuevo. Una vista general de la arquitectura del sistema es presentada en su correspondiente diagrama de componentes, en la Figura 4.2. La arquitectura en capas cuenta con las ventajas de encapsular las labores de cada capa, ocuparse de diferentes peticiones al mismo tiempo, securizar cada capa y modificar la implementación de cada capa sin alterar el funcionamiento de las no adyacentes. Del mismo modo, dicha arquitectura software cuenta con las desventajas de no ser la más eficiente (cada petición atraviesa dos veces cada capa) o que todas las capas sean críticas para el desempeño del sistema (no es especialmente resistente a caídas o fallos).

Capítulo 5

Diseño de interfaz

Esta sección recoge todo lo referente al desarrollo del diseño y la maquetación de los elementos del *frontend* de la aplicación Web. Por no ser esta tarea la labor principal del proyecto; el equipo busca, sin descuidar la apariencia, lograr una interfaz sencilla y funcional que permita a un usuario enviar y recibir sus consultas y leer algunos ejemplos de uso del sistema. En una iteración posterior del proceso de desarrollo, se plantea e incluye, además, la funcionalidad de modificar desde la interfaz Web el contexto que recibe el sistema, de modo que una persona con experiencia en LLM y ASP pudiera, si lo desease, alterar el funcionamiento del programa.

5.1 Tecnologías usadas

5.1.1 HTML, CSS y Javascript

El Lenguaje de Marcas de Hipertexto (HTML) es un lenguaje de marcas con el propósito de estructurar contenido, y es usado de forma casi universal en el contexto Web. Por ello, su uso en el proyecto resulta de difícil evasión. Una de las peculiaridades de este lenguaje reside en la capacidad de referenciar unas páginas con otras (a dichos enlaces se les denomina hipertexto).

Las Hojas de Estilo en Cascada (CSS) son un lenguaje empleado para dotar de estilos y cambiar el aspecto de archivos previamente estructurados (con, por ejemplo, HTML o Lenguaje de Marcas Extendible (XML)). Su uso conjunto con HTML está muy extendido en el ámbito del diseño Web, por su utilidad a la hora de determinar la parte visual de una página.

JavaScript es en lenguaje de programación basado en prototipos e interpretado, especializado y usado principalmente para el desarrollo de páginas Web interactivas. Tiene la labor de definir el comportamiento y un gran número de funcionalidades dinámicas de una página Web. Es la tecnología encargada de, por ejemplo, reproducir contenido audiovisual o reaccio-

nar a eventos del usuario.

En conjunto, las tres tecnologías tienen en la actualidad el reconocimiento por parte de un gran número de desarrolladores de todo el mundo, siendo percibidas por una amplia mayoría como herramientas casi ineludibles en el campo del diseño Web.

5.1.2 Bootstrap

Bootstrap es un *framework* de código abierto dirigido al desarrollo Web. Aúna HTML, CSS y JavaScript, haciendo el papel de librería integral entre ellos. Añade un gran número de funcionalidades y simplifica numerosos procedimientos, abstrayendo y facilitando el proceso para el desarrollador.

5.2 Pantallas de la interfaz

La interfaz se compone por tres pantallas individuales: la pantalla ”Chat” (1), encargada de todas las funcionalidades principales del sistema (la petición de resolución de puzzle, la selección del puzzle a resolver o la modificación del contexto); la pantalla ”Ayuda” (2), con explicaciones de los puzzles implementados y la presentación de ejemplos de uso, y la pantalla ”Acerca De” (3), con información de contacto y autoría.

5.2.1 Chat

La pantalla ”Chat” (Figura 5.1) cuenta con los elementos necesarios para llevar a cabo los casos de uso definidos para el sistema. Como pieza central de la interfaz se encuentra el recuadro de texto (Figura 5.2), donde el usuario interactuará directamente con el sistema. En ella, puede encontrarse de un lado o del otro una colección de mensajes de texto o de imagen, donde a la derecha se encuentran las peticiones del usuario y a la izquierda las respuestas del sistema, emulando el funcionamiento de una aplicación de mensajería. Las imágenes resultantes son también un elemento interactuable, y puede el usuario acceder a una vista ampliada (Figura 5.3) tras hacer clic en una de ellas en su respectivo mensaje en el chat. En el propio recuadro de chat, además, el usuario puede elegir uno de los puzzles implementados (actualmente, *Einstein* o *Comensales*) mediante un selector vertical (Figura 5.4).

Como último elemento relevante de la pantalla se encuentran las entradas de textos correspondientes a la pestaña de ”Opciones avanzadas” (figura 5.5), donde un usuario con de-

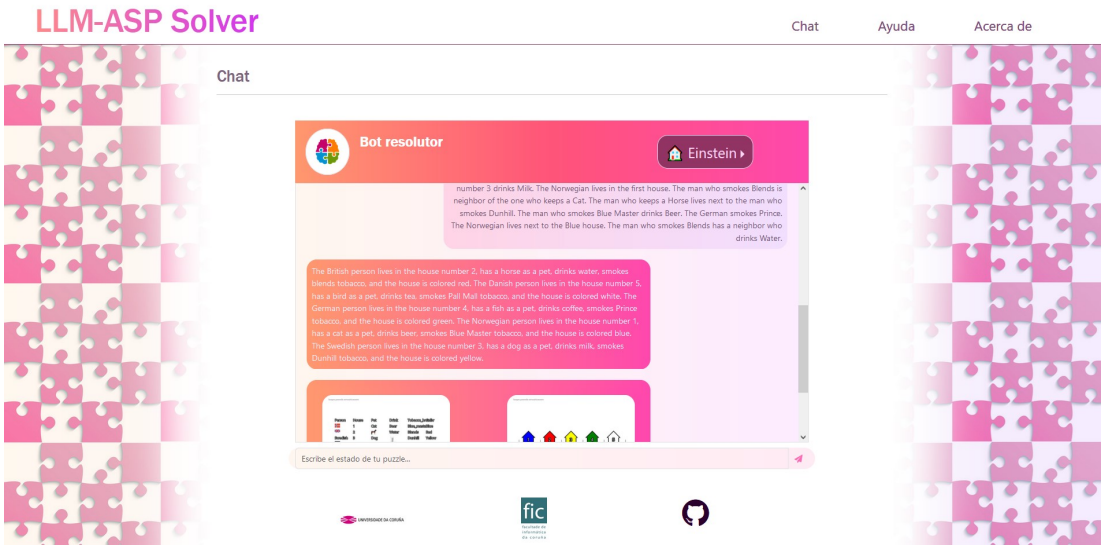

Figura 5.1: Pantalla ”Chat”.

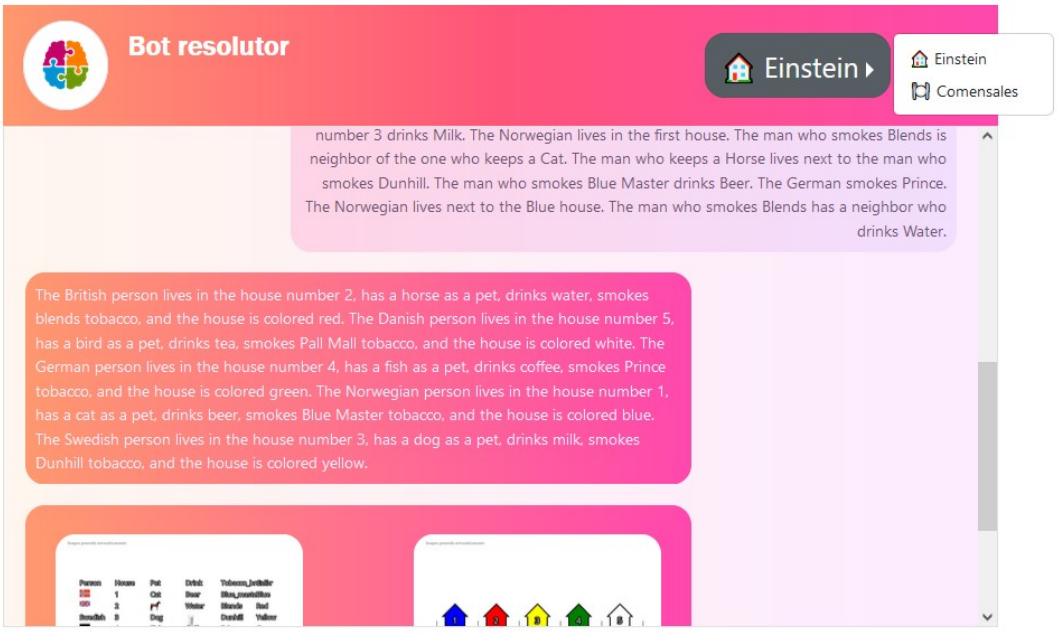

Figura 5.2: Ampliación sobre el recuadro de chat, con el *dropdown* de selección de puzzles desplegado.

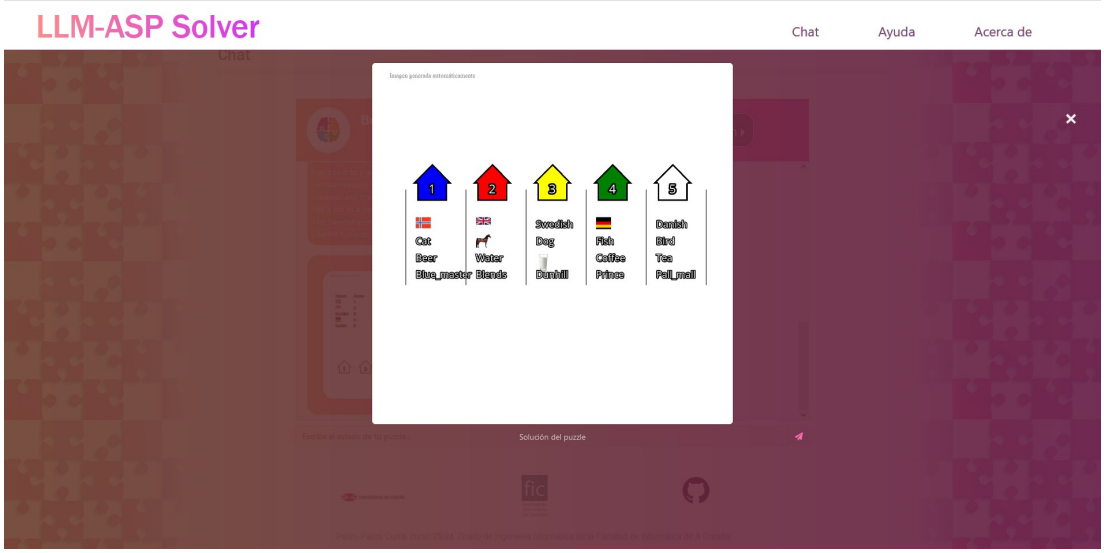

Figura 5.3: Ventana de imagen ampliada, resultado de hacer clic en una imagen resultante de una petición.

Figura 5.4: *Dropdown* para la selección de puzzle a resolver.

terminada experiencia podría variar las entradas del modelo de lenguaje a bajo nivel y, por lo tanto, optimizar también las salidas del sistema bajo su propia responsabilidad.

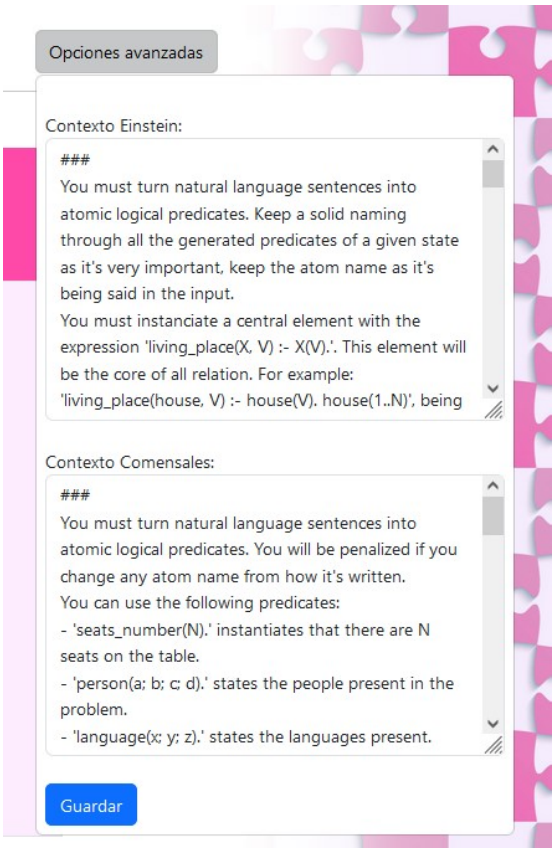

Figura 5.5: Pestaña de opciones avanzadas.

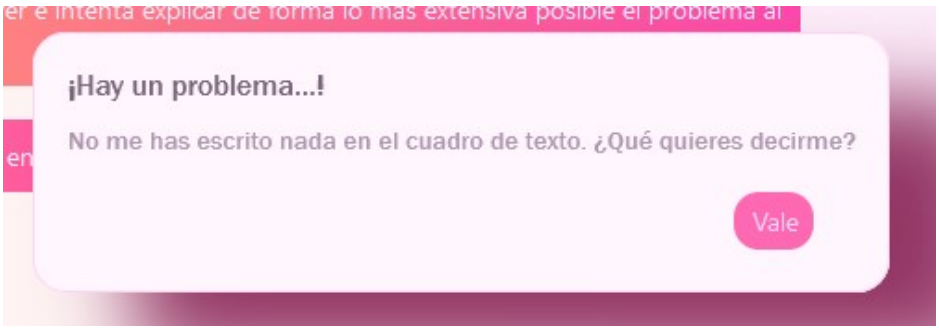

Figura 5.6: Diálogo indicando la no selección de un puzzle.

Cabe mencionar que en esta misma pantalla ”Chat” pueden ser lanzados diversos diálogos como el de la Figura 5.6, fruto de interacciones indebidas (como enviar una petición sin

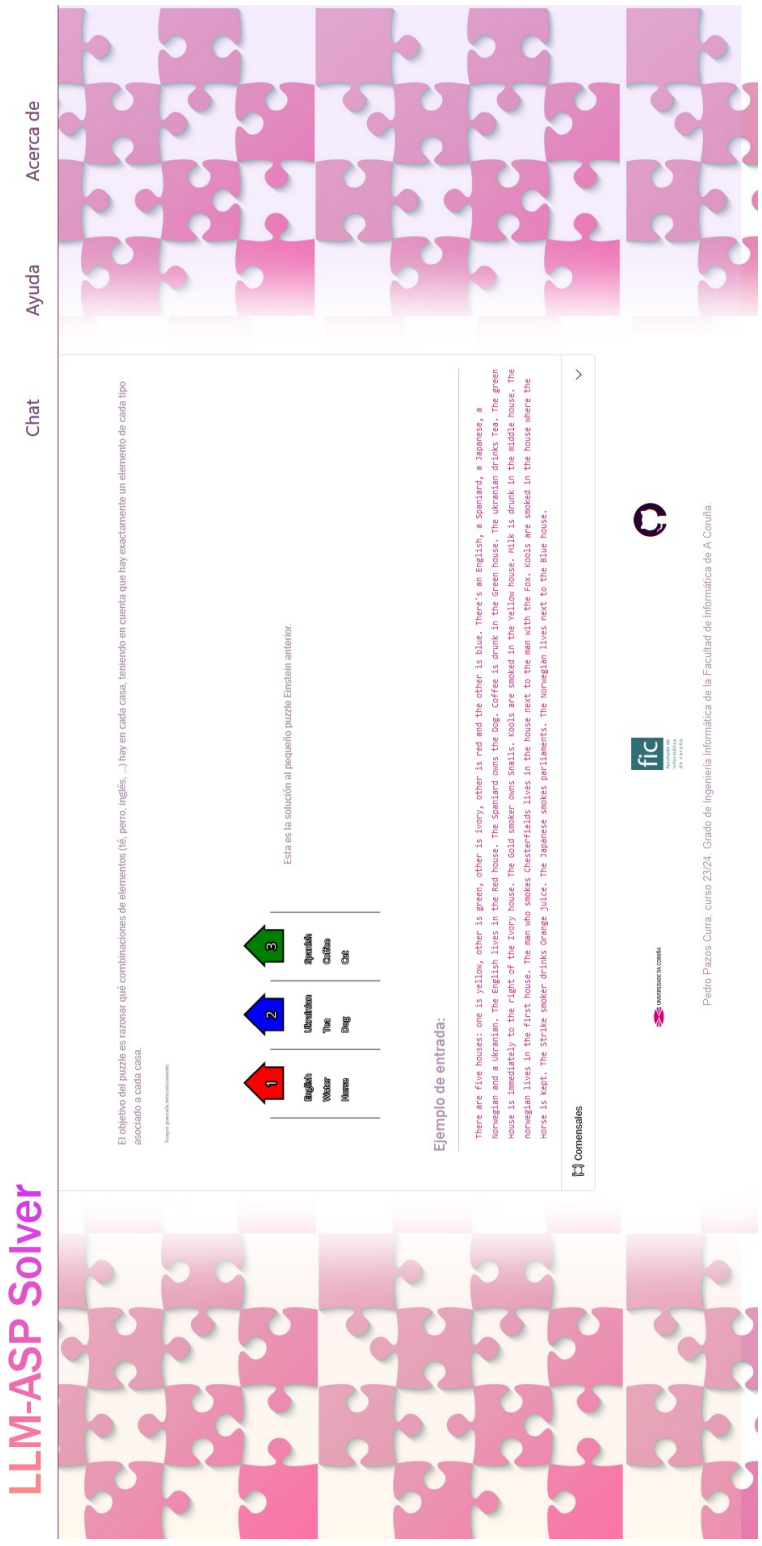

Figura 5.7: Pantalla ”Ayuda”

Figura 5.8: Pantalla "Acerca De"

seleccionar un puzzle) o de errores inesperados devueltos por el servidor (fallo del servidor, error en la Application Programming Interface (API), etcétera).

5.2.2 Ayuda

La pantalla ”Ayuda” apoya a los neófitos en el uso del programa a entender con la menor fricción posible el funcionamiento del sistema. Dota a través de imágenes y texto de la información necesaria para comprender las reglas de los puzzles implementados. Además, apoyándose en la salida del módulo gráfico del sistema, explica los estados inicial y final de una posible petición. Por último, da un ejemplo de una petición que el usuario podría realizar, para que el mismo pudiera copiarla y pegarla para hacer uso de ella a modo de primera prueba del sistema. Esta pantalla es presentada en la Figura 5.7.

5.2.3 Acerca de

Como se menciona previamente, la pantalla ”Acerca de” consta con una función prácticamente anecdótica, dejando un espacio de mención de los autores y correo de contacto, así como enlaces e información sobre el proyecto. Una captura de dicha pantalla es expuesta en la Figura 5.8.

5.3 Comunicación con el servidor

La interfaz cuenta con diversas acciones cuya reacción implica lanzar una petición al servidor (los detalles específicos de dichas peticiones son abordadas en el Capítulo 6), como es práctica común en la industria. Diversos ejemplos de estas acciones son, prominentemente, el envío de un mensaje o la petición de diversos de los elementos que el servidor salvaguarda de forma estática, esto es, los recursos: imágenes, contexto dado a los puzzles o los propios códigos HTML usados en las pantallas. Dichas peticiones se hacen mediante la herramienta usada Express.js (cuya función es aludida en el próximo Capítulo 6), que realiza la labor mediante la incorporación de la técnica Asynchronous JavaScript and Xml (AJAX).

A continuación, se presenta un ejemplo del código JavaScript de una de las peticiones desde el lado del *frontend*; el envío del mensaje, concretamente. En la interfaz de usuario, el evento de pulsar el botón que envía el mensaje lanza la función *userEnviarMensaje()*. Esta función coloca el mensaje del usuario en la pantalla y a su vez lanza el método *chatbotEnviarMensaje()*, que ejecuta la función *peticionProcesaMensaje()* y se encarga también de recibir su

resultado, manejar casos de error, y en definitiva todos aquellas labores del lado del *chatbot* en la conversación. Tras todo el proceso, es esta última función (cuyo código es expuesto a continuación) la que finalmente realiza la petición al servidor. En este caso concreto, se hace una petición POST a la ruta */procesa-mensaje*, pues es la interfaz expuesta por parte del servidor para resolver esta petición particular.

```
async function peticionProcesaMensaje(mensaje){

  return fetch('/procesa-mensaje', {
    method: "POST",
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      message : mensaje,
      puzzle : selectedPuzzle
    })
  }).then(response => { // Caso exitoso
    return response.json();

  }).catch(error => {   // Caso de error -> Aviso
    mostrarDialogo("...Ha habido un error enviando tu mensaje al
    servidor, lo sentimos.\n\n" + error);

  });

}
```

En el capítulo próximo (Capítulo 6) se hace hincapié en la recepción de peticiones, por lo que no ahonda el presente Capítulo en seguir el flujo expuesto.

Capítulo 6

Servidor

Es discutido en el actual Capítulo el desarrollo de la primera capa del backend del programa, correspondiente al servidor.

6.1 Tecnologías usadas

6.1.1 Node.js y Express.js

Node.js es un entorno de ejecución de Javascript de código abierto y orientado a lanzar código en la parte del servidor. Provee a los creadores numerosas herramientas para construir sus proyectos. De entre estas herramientas, el sistema se beneficia de un *framework*, Express.js, que dota al desarrollador de una plataforma sobre la que lanzar un sistema Web. Express.js presenta al usuario una interfaz sencilla con la que dar vida a determinadas partes de un servidor. En concreto, se busca con el útil conseguir el objetivo de poner en funcionamiento una API REpresentational State Transfer (REST), encargada de recibir las peticiones en forma de acciones por parte del usuario (pulsar el botón de enviar mensaje, cambiar de pantalla en el menú, etcétera) y ejecutar la lógica del programa sobre ellas para tener la capacidad de enviar una respuesta a esa petición, mutando la pantalla del usuario y manteniendo el estado en el lado del servidor.

6.1.2 Implementación

Usando las tecnologías previamente mencionadas, es implementada una capa de servidor REST que escucha peticiones GET, DELETE, POST o UPDATE en un puerto interno del equipo mediante el método Asynchronous JavaScript and Xml (AJAX) facilitado por Express.

Son definidas diversas rutas que sirven como elemento de entrada para las peticiones del

frontend. Estas rutas, denominadas nodos o *endpoints*, están definidas mediante una ruta que define su cometido; por ejemplo, cuando se recibe un GET a la ruta */ayuda*, se envía el archivo HTML de la pantalla de ayuda. Para ejemplificar en profundidad esta interfaz expuesta, se detalla a continuación el código JavaScript de la ruta */procesa-mensaje*, encargada de recibir los mensajes de los usuarios.

```
1
2 // Ruta que recibe POST en '/procesa-mensaje'
3 app.post('/procesa-mensaje', (req, res) => {
4
5   // Coge mensaje del cuerpo del JSON + LOG
6   const msg = req.body.message;
7   const puzzle = req.body.puzzle;
8   console.log("LOG: POST /procesa-mensaje: \"%s\" para puzzle: %s
      ", msg, puzzle);
9
10  // Se procesa el mensaje recibido (nota: es asíncrono) y se
      responde
11  procesaMensaje(msg, puzzle).then((valor) => {
12    res.send(JSON.stringify(valor));
13  });
14
15 });
```

La ruta */procesa-mensaje* recibe tanto el mensaje de entrada como el puzzle escogido en la interfaz de usuario. Lanza entonces la función *procesaMensaje()*, encargada de lanzar el script del proceso del sistema neurosimbólico (escrito en Python). Una vez finalizada la tarea, recoge la salida de esta capa y la devuelve hacia la ruta del servidor que recibió la petición, devolviéndola a su vez al *frontend*.

```
1 /* Función asíncrona procesaMensaje
2 */
3 async function procesaMensaje(mensaje, puzzle){
4
5   const respuesta = await ejecutaScript(path.join(__dirname,
      '../py/proceso.py'), mensaje, puzzle);
6   return(respuesta);
7
8 }
```

Aclarado el funcionamiento del servidor y su papel en el flujo de las peticiones en el *backend*, resta profundizar en el siguiente Capítulo 7 el trabajo realizado en el dominio del sistema neurosimbólico.

Capítulo 7

Sistema neurosimbólico

7.1 Introducción

Es propuesto en el Capítulo introductorio (Capítulo 1) la posible implementación del sistema neurosimbólico a través de 4 etapas lógicamente independientes que dividen las tareas a realizar. En la presente sección son desglosadas con detalle cada una de las fases de dicho sistema. Las partes que lo componen son las siguentes: generación de predicados ASP partiendo de LN (1), la búsqueda de soluciones usando el *solver* Clingo (2), la descripción de las soluciones halladas (*answer sets*) a LN (3) y la representación gráfica de las soluciones obtenidas (4).

En cuanto al código ASP, el equipo diseña una sintaxis reducida para ambos puzzles con el objetivo de maximizar la adherencia del LLM a la vez que garantizar que el estado del puzzle es explicado en completitud. Otra variable deseada en el diseño de este lenguaje común es la flexibilidad para poder modificar parámetros del puzzle en la medida de lo posible sin alterar su lógica y mantener un buen funcionamiento del mismo (por ejemplo, emplear barcos o caravanas en lugar de casas para el puzzle *Einstein*). Se presentan, además, dos átomos de representación (*show_graphic* y *show_description*), independientes del puzzle, para que el usuario pueda indicar si desea recibir la representación gráfica, de texto o ambas. La tabla 7.1 recoge los predicados escogidos por el equipo.

En lo referente al modelo de lenguaje, es planteada en primera instancia la idea de manejar el LLM ejecutándolo en una máquina en propiedad del equipo de desarrollo. Para este fin, se hace uso de Oobabooga[25], una herramienta *software* que permite usar, cambiar o ajustar los hiper-parámetros de cualquier LLM abierto disponible en la página *Hugging Face* (e importar los obtenidos por otros medios), contando además con documentación suficiente e interfaz web que abstrae y facilita el trabajo de desarrollo. Tras ejecutar diversos modelos en el equipo

disponible, escrutando todo el rango en cuanto a capacidad inferencial se refiere, se observa que el *hardware* disponible no permite los modelos con una capacidad superior a la de aquellos 7B, por lo que tras una comprobación, el equipo puede aseverar que el material disponible no es óptimo ni siquiera para modelos de la talla mínima. En concreto, y por disponer de 6 Gigabytes (GB) de Video Random-Access Memory (VRAM) en el equipo usado para las pruebas, no se alcanzan siquiera los requisitos mínimos impuestos por un modelo 7B convencional. Por ejemplo, *LLaMA 7B*, uno de los LLM lanzados por Meta, exige, para llevar a cabo una inferencia con total precisión, 28 GB de RAM en la tarjeta de vídeo (VRAM). Este cálculo se hace a partir de multiplicar cada parámetro del modelo, almacenado en 4 bytes por los $7 \cdot 10^9$ de parámetros del modelo. Por este motivo, el único modo por el que los modelos pueden ejecutarse en el sistema es a través de la llamada cuantización, un método a través del cual se reduce la precisión de las inferencias del modelo, para permitir su ejecución en medios más humildes. Tras intentar realizar inferencias usando este método, se comprueba que tampoco se consigue de esta forma subsanar los problemas de rendimiento tanto en tiempo de ejecución como en capacidad inferencial. Para abordar la situación, el equipo estima beneficioso hacer uso de una API externa de inferencias de modelos de lenguajes llamada AwanLLM, fundada en 2023 y con las ventajas de tener un plan gratuito flexible con 200 peticiones diarias, diferentes modelos disponibles y ventanas de contexto más que suficientes (variables según el modelo, pero siempre por encima de 8192 tokens); con la superioridad principal de no requerir un equipo potente para realizar las inferencias del modelo. De entre los modelos disponibles en la herramienta, se propone el uso de *Meta-Llama-3-8B-Instruct*, uno de los modelos de la línea de LLM abiertos de Meta (anteriormente Facebook), la conocida tecnológica americana.

Respecto al flujo procesal que aúna los cuatro subsistemas, en los cuales se ahonda en las consecuentes subsecciones, cabe remarcar que se idea una arquitectura líder-trabajador donde los subsistemas realizan su labor y envían su respuesta al proceso central para que continúe manejándola. Esta estructura queda patente en el diagrama de componentes del sistema neurosimbólico en detalle, expuesto en la Figura 7.1. Además, todos los mensajes entre los subsistemas y el proceso central siguen un formato similar compuesto por una tupla de un código de estado y el mensaje u objeto devuelto.

7.2 Generación de predicados ASP (*NL_to_ASP*)

Como primera fase del proceso inferencial, se propone un módulo encargado de, mediante el LLM, transformar frases en LN en predicados lógicos ASP. Para ello, se hace una llamada a la API del modelo, añadiendo en la ventana de contexto un conjunto de directrices

Puzzle	Predicado	Descripción del predicado
Einstein	type(T, V) :- T(V)	Inicializa un tipo T i.e ”type(color, V) :- color(V)”
	T(A) o T(A; B; C)	Instancia uno o más átomos de tipo T
	living_place(P, V) :- P(V)	Inicializa un elemento central P.
	P(1..n)	Instancia n elementos centrales P indicados previamente con *living_place*.
	same_place(X,Y)	El átomo X está agrupado con el átomo Y (pueden ser también números siempre que se haya inicializado el elem. central)
	next_to(X,Y)	El átomo X y el átomo Y están en elems. centrales adyacentes.
	left(X,Y)	El átomo X está a la izquierda del átomo Y.
	right(X,Y)	El átomo X está a la derecha del átomo Y.
Comensales	*seats_number(N)*	Declara el número de asientos del juego.
	person(A) o person(A; B; C)	Instancia las personas presentes.
	language(X; Y; Z)	Instancia los lenguajes presentes.
	speaks(P, L)	Indica que la persona P habla el lenguaje L.
	next_to(X,Y)	Indica que la persona X debe estar al lado de una persona Y o una persona que hable el lenguaje Y (Y puede ser o bien una persona o un lenguaje).
	not_next_to(X,Y)	Indica que la persona X no puede estar al lado de una persona Y o una persona que hable el lenguaje Y (Y puede ser o bien una persona o un lenguaje).
	opposite(X,Y)	Indica que las personas X e Y deben estar sentadas en lados opuestos de la mesa.
– (Común)	show_graphic	Mostrar solución gráfica
	show_description	Mostrar descripción de solución en texto

Tabla 7.1: Sintaxis de predicados escogidos como lenguaje común para la entrada de la sección ASP

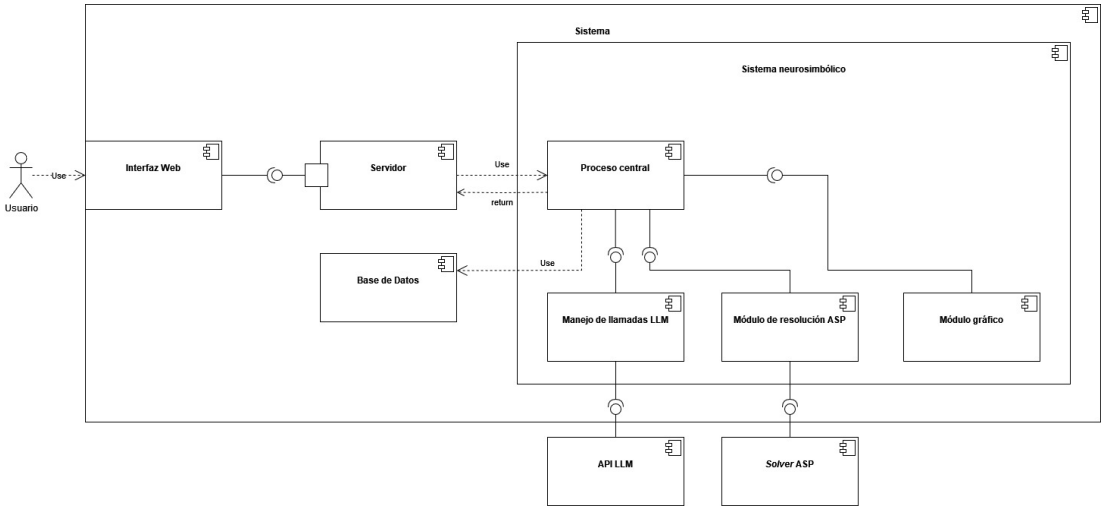

Figura 7.1: Diagrama de componentes con detalles de la estructura del sistema neurosimbólico ampliados.

de cómo debe actuar, qué no debe hacer y una serie de ejemplos de entradas y salidas esperadas. Está técnica es conocida como *in-context learning*, y el equipo accedió a su uso debido a la sencillez de implementación; pero también, y en especial, por la falta de una cantidad significativa de ejemplos de transformaciones LN-ASP (y viceversa) etiquetadas; atributo de obligado cumplimiento para el caso de algunas aproximaciones alternativas (comentadas en el apartado 2.2.2).

A continuación es expuesto un fragmento de la ventana de contexto aportada para el puzzle *Einstein*:

```
###
You must turn natural language sentences into atomic logical predicates. Keep a solid naming through all the generated predicates of a given state as it's very important, keep the atom name as it's being said in the input.
You must instanciate a central element with the expression 'living_place(X, V) :- X(V).'. This element will be the core of all relation. For example: 'living_place(house, V) :- house(V). house(1..N)', being N the number of houses. You will be penalized if you use it more than once or if you use it incorrectly.
You can use the following predicates:
- 'same_place(X,Y).': atom X and the atom Y are on the same place or grouped together, for example 'John lives in the house number 3' would translate to 'same_place(john, 3).', while 'Water is drunk in the house where Camel is smoked' would be 'same_place(water, camel).'
- 'next_to(X, Y)': X and Y atoms are one next to another, as 'Jose and Marta are neighbors' would be 'next_to(jose, marta).' or 'The Dog is in the next house where the Cat owner is.' would be 'next_to(dog, cat).'.
- 'left(X, Y).': atom X is strictly to the left of atom Y. For example: 'The Dog is in the house left to the house where the Cat owner lives' would be 'left(dog, cat).', and 'Mary lives to the left of the Blue house' would be 'left(mary, blue).'
- 'right(X, Y).': atom X is strictly to the right of atom Y. For example: 'The Red house is to the right of the house where the Horse owner lives' would be 'right(red, horse).' and 'Tim lives to the right of the house of the Pall Mall smoker.' would be 'right(tim, pall_mall)'
- 'show_graphic.': this predicate indicates that the user wants to get a visual, graphic representation. For example, if the input says something like 'Represent it visually' or 'Draw the solution', then 'show_graphic.' must be added to the output.
```

```
- 'show_description.': this predicate indicates that the user wants to get a text description for the solution. For example, if the input says something like 'Explain the result' or 'Describe the solution', then 'show_description.' must be added to the output.
In the case of indicating a place number, you must always use the number '1,2,3' when provided a number 'one, two, three' or an ordinal 'first, second, third'. For example: 'The dog lives in the first house' is 'same_place(dog, 1).', and 'The dog lives in the house number two' is also 'same_place(dog, 2).'.
Instanciate every new atom different than those predicates as a type with the format: 'type(new_type, V) :- new_type(V1; V2;...; Vn).'. For example: 'type(pet, V) :- pet(V). pet(dog; cat; horse).' defines the pets Dog, Cat and Horse, and 'type(color, V) :- color(V). color(red; blue; green).' defines colors Red, Blue and Green.
You will be penalized if you write anything in natural language. You will be penalized if you make any kind of note or clarification. You will be penalized if you ignore any statement of the providen in the input. You will be penalized if you're verbose and convoluted. Complete only the last iteration.
Given the following examples, process only the last input. You will be penalized if your output is non ASP parsable.
###
INPUT: The house colours are White, Red, Green and Yellow. There's a German, a Polish and an English. The person living in the Red house is the neighbor of the owner of the Green house.
OUTPUT: living_place(house, V) :- house(V). type(color, V) :- color(V). color(white; red; green; yellow). person(german, polish, english). next_to(red, green).
IN: The owner of the Dog lives next to the owner of the Red house. The Red house owner lives in the first house. There are 9 houses.
OUT: living_place(house, V) :- house(V). type(pet, V) :- pet(V). pet(dog). type(color, V) :- color(V). color(red). next_to(dog, red). same_place(red, 1). house(1..9).
IN: There are ten houses. The Swedish lives in the last one. The Spanish lives in the first one. The German lives in the third one.
OUT: house(1..10). living_place(house, V) :- house(V). person(swedish; spanish). same_place(swedish, 10). same_place(spanish, 1). same_place(german, 3).
(...)
```

En lo que al contexto dado se refiere, se procura la utilización de diversas prácticas comunes del *prompt-engineering* recogidas en la bibliografía revisada; como el presentar al modelo un rol de especialista humano, la utilización de lenguaje serio y rígido, casi marcial; la imposición de castigos ficticios a salidas no deseadas o la presentación de las instrucciones de forma lo más sintetizada e informativa posible. Además, y lo que es más importante, se presentan multitud de ejemplos de funcionamiento (en su mayoría omitidos por visibilidad en el fragmento descrito, pero mostrados al completo en el documento anexo A).

Tras recibir la respuesta del modelo de lenguaje y antes de permitir que continúe en el flujo del sistema, se efectúa un filtrado dividido en tres fases. En primer lugar (1), un post-procesado estático con el objetivo de rescatar la salida de comportamientos no deseados reconocidos por el conocimiento experto del equipo de desarrollo. Por ejemplo, se observa que debido a su inherente incomprensión del lenguaje ASP, confunde muy esporádicamente alguna de sus reglas sintácticas. Ante casos como estos se lleva a cabo un filtrado trivial y estático como el expuesto a continuación.

```
    # Preprocesado de caracteres extraños que a veces el LLM mete por formato. También estructuras ilegales en el programa ASP, por mejorar resultados marginalmente.
    if (salida_llm.find("`") != -1):
        salida_llm = salida_llm.replace("`", "")
```

```
    if (salida_llm.find("),") != -1):
        salida_llm = salida_llm.replace("),", ").")
    if (salida_llm.find("type(person, V) :- person(V).") != -1):
        salida_llm = salida_llm.replace("type(person, V) :-
    person(V).", "")

    salida_llm = salida_llm.strip()    # A veces se olvida de poner
    el punto en el último predicado.
    if (salida_llm[-1] != "."):
        salida_llm += "."
```

En segundo lugar (2), se realiza un filtrado laxo mediante Regular Expression (REGEX), con la pretensión de detectar subconjuntos de la salida que sí sean compatibles con sintaxis ASP y separarlos de los que no. Esta prueba viene dada por la tendencia observada del modelo de lenguaje a introducir sus respuestas con una apertura en la línea de "*Here is the output:*", así como finalizarla con un texto similar a "*I hope this was of help*". Una vez filtrado el texto, y de encontrar un fragmento ASP-válido, se recorta sólo este y se continúa a la siguiente fase del postprocesado.

En última instancia (3), se efectúa un filtro estricto, otra vez con REGEX. En este caso, en lugar de buscar un subconjunto ASP, comprueba si la salida cumple estrictamente con la sintaxis diseñada para el puzzle particular. Por la rigidez de la prueba, el código REGEX es específico para cada puzzle, al contrario que las anteriores fases. Se detalla a continuación un fragmento de los dos últimos pasos del procedimiento, junto con los códigos REGEX propuestos para la solución.

```
# Filtros REGEX
ASP_REGEX_LIGERO = r"(
([a-z\_]+\(([a-z\_]+\,\s?[V])\)\s:-\s[a-z\_]+\([A-Z]\)\.?\s?)
|(\s?[a-z\_]+\([0-9]+\.\.[0-9]+\)\.?\s?)
|(\s?[a-z\_]+\((\s?[a-z\_]+\;?\s?)+\)\.?\s?)
|(\s?[a-z\_]+\(([a-z\_0-9]+\s?\,?\s?)+\)\.?\s?)
|(\s?show_graphic.\s?|\s?show_description.\s?)
)+"

ASP_REGEX_ESTRICTO_EINSTEIN = r"^(
(living_place\(([a-z\_]+\,\s?[V])\)\s:-\s[a-z\_]+\([V]\)\.?\s?)
|(type\(([a-z\_]+\,\s?V)\)\s:-\s[a-z\_]+\(V\)\.?\s?)
|(\s?[a-z\_]+\([0-9]+\.\.[0-9]+\)\.?\s?)
|(\s?[a-z\_]+\((\s?[a-z\_]+\s?\;?\s?)+\)\.?\s?)
|(\s?(living_place|left|right|next_to|same_place)\(([a-z\_0-9]+\s?\,?\s?)+\)\.?\s?)
|(\s?show_graphic.\s?|\s?show_description.\s?)
```

```
)+$"

(...)

# Aplicación de REGEX laxo: búsqueda de grupos que acepten sintaxis ASP
match_regex_ligero = re.search(ASP_REGEX_LIGERO, salida_llm, flags=0).group()

# Aplicación de REGEX estricto: validación de sintaxis propuesta para el puzzle
try:
    if(puzzle == "Einstein"):
        match_regex_estricto_einstein = re.search(ASP_REGEX_ESTRICTO_EINSTEIN, match_regex_ligero, flags=0).group()
        if match_regex_estricto_einstein == match_regex_ligero:
            return([0, match_regex_estricto_einstein])  # OK
        else:
            (...)
except:
    (...)
```

7.3 Resolución de ASP (*resolver_ASP*)

En segundo lugar del flujo procesal se encuentra el apartado simbólico de la solución. Para alcanzarla, y como se referencia en el Capítulo 2, se hace uso de la interfaz de Clingo para Python, abstrayendo y facilitando al equipo el uso del aparato lógico del Answer Set Programming.

Este módulo recibe la salida de la fase anterior (7.1), y su objetivo es recibir código ASP que describe el estado del puzzle y enviar por salida una (y sólo una, por más que pudiera tener el programa lógico) solución o *answer set*. Como se destaca en el Capítulo introductorio (1), el sistema alberga de forma estática las reglas del puzzle en cuestión puesto a que su generación no queda abarcada en el alcance del proyecto. A continuación, se hará constar el código ASP con las reglas de ambos puzzles, cuya creación sí forma parte del trabajo realizado:

En primer lugar, se exponen a continuación el código ASP de las reglas de *Einstein*:

```
% Filtrado previo de entradas erróneas en cuanto a lógica de puzzle
:- same_place(X,Y), person(X), not type(_,Y).
```

```
:- same_place(X,Y), not type(_,X), person(Y).
:- same_place(X,Y), not type(_,X), not type(_,Y).
:- same_place(X,Y), not person(X), not type(_,X).
:- same_place(X,Y), not person(Y), not type(_,Y).
:- same_place(X,Y), person(X), person(Y).
:- not living_place(_,_).

% Reglas para determinar cosas de un elem. central (oculto a llm)
place_of(X,N) :- has(P,_,X), has(P,Y,N), X!=N, living_place(Y,N).
place_of(P,N) :- has(P,Y,N), living_place(Y,N).
type(P, V) :- living_place(P,V).

% Generar soluciones: todos los elementos inicializados asignados a
    una persona
1 { has(X,T,V): person(X) } 1 :- type(T,V).
:- has(X,T,V), has(X,T,V'), V!=V'.

%%%%%%%%%%%%%%%%%%%%%%   Interfaz de LLM   %%%%%%%%%%%%%%%%%%%%%%

% same_place (X, nº ) / same_place (nº , X)
:- same_place(X,N), not person(X), living_place(_,N), has(P,_,N),
    not has(P,_,X).
:- same_place(N,X), not person(X), living_place(_,N), has(P,_,N),
    not has(P,_,X).
:- same_place(P,N), person(P), living_place(L,N), not has(P,L,N).

% same_place(X,Y).
:- same_place(X,Y), place_of(X,H), place_of(Y,H'), H!=H'.
1 {has(P,T,X): person(P)} 1 :- same_place(X,Y), type(T,X).
1 {has(P,T,Y): person(P)} 1 :- same_place(X,Y), type(T,Y).

% next_to(X, Y)
:- next_to(X,Y), place_of(X,H), place_of(Y,H'), |H-H'| != 1.
:- next_to(X,Y), X==Y.
:- next_to(X,Y), same_place(X,Y).
next_to(Y,X) :- next_to(X,Y).

% left(X,Y), right(X,Y)
:- left(X,Y), place_of(X,H), place_of(Y,H'), X!=Y, H!=(H'-1).
:- right(X,Y), place_of(X,H), place_of(Y,H'), X!=Y, H!=(H'+1).
next_to(X,Y), right(Y,X) :- left(X,Y).
next_to(X,Y), left(Y,X) :- right(X,Y).

#show has/3.
```

En segundo lugar, se presenta el código ASP de las reglas del puzzle de *Comensales*:

```
%% Filtrado de entradas incorrectas
:- next_to(X,Y), not person(X), not language(X).
:- next_to(X,Y), not person(Y), not language(Y).
:- next_to(P,P'), P == P'.
:- not_next_to(X,Y), not person(X), not language(X).
:- not_next_to(X,Y), not person(Y), not language(Y).
:- not_next_to(P,P'), P == P'.
:- speaks(A,L), person(A), not language(L).
:- speaks(A,L), not person(A), language(L).
:- speaks(A,L), not person(A) , not language(L).
:- speaks(A,L), speaks(A,L'), L!=L'.
:- opposite(P,P'), not person(P).
:- opposite(P,P'), not person(P').
:- opposite(P,P'), P == P'.

%% seats_number(N) -> seat(1..N)
seat(1..N) :- seats_number(N).

neighbor(X,Y) :- person(X), person(Y), seated(X,S), seated(Y,M),
    seats_number(S), X !=Y, S <= 3.
neighbor(X,Y) :- person(X), person(Y), seated(X,S), seated(Y,M),
    seats_number(S), X !=Y, M != S,  M\(S-2) == 1.
neighbor(X,Y) :- person(X), person(Y), seated(X,N), seated(Y,S),
    seats_number(S), X !=Y, N != S,  N\(S-2) == 1.
neighbor(X,Y) :- person(X), person(Y), seated(X,N), seated(Y,M),
    seats_number(S), X !=Y, N != S, M != S,  |N- M| == 1.
neighbor(Y,X) :- neighbor(X,Y), X != Y.

%% next_to(X,Y)
:- next_to(X, Y), person(X), person(Y), not neighbor(X, Y).
:- next_to(X, L), seats_number(S), S > 3, person(X), language(L),
    person(Y), speaks(Y,L), not neighbor(X,Y).
:- next_to(L, X), seats_number(S), S > 3, person(X), language(L),
    person(Y), speaks(Y,L), not neighbor(X,Y).

%% not_next_to(X,Y)
:- not_next_to(X, Y), person(X), person(Y), neighbor(X, Y).
:- not_next_to(X, L), person(X), language(L), speaks(Y,L),
    neighbor(X,Y).
:- not_next_to(L, X), person(X), language(L), speaks(Y,L),
    neighbor(X,Y).

%% opposite(X,Y)
:- opposite(X,Y), person(X), person(Y), seated(X,N),
    seats_number(SIZE), N == SIZE/2,  not seated(Y, 1).
```

```
:- opposite(X,Y), person(X), person(Y), seated(X,N),
    seats_number(SIZE), N != SIZE/2, not seated(Y,
    ((N+(SIZE/2))\SIZE)).

%% Átomo final seated(X,n).
1 {seated(P,N) : person(P)} 1 :- seat(N).
:- seated(P,N), seated(P,M), N!=M.
:- seated(P,N), seated(P',N), P!=P'.

#show seated/2.
```

En ambos puzzles, y como es habitual en un código ASP, se generan todas las posibles combinaciones de resultados con la línea:
$1\{predicado\_a\_generar(p_1, p_2, ..., p_n) : person(P)\}1 : -cond\_1, cond\_2, ..., cond\_m.$

Leído como "dadas las m condiciones, genera para cada persona 1 y sólo 1 predicados a generar". Esto resulta en, por ejemplo, que existiendo un asiento *seat(1)* y una persona *person(francisco)*, se genere en el progama el átomo *seated(francisco, 1)*, siempre y cuando se cumplan también el resto de reglas propias del puzzle.

Con los códigos de reglas previamente declarados, el código ASP de entrada proveniente de la fase anterior y el puzzle escogido, se procede a concatenar los programas ASP y a ejecutar el proceso de *grounding* y *solving* mediante la API de Clingo (2.1). El resultado, de existir una solución al programa lógico, consistirá en el conjunto de átomos mínimos que acuerden todas las reglas impuestas. Esto es, en el puzzle *Einstein*, un conjunto de predicados *has(persona, tipo, elemento)*, indicando todos los elementos agrupados a las personas presentes; mientras que en *Comensales* la solución se presentará a través del átomo *seated(persona, número)*, indicando el asiento tomado por cada persona presente. Esto no es todo; además, la solución hallada presenta los átomos de representación *show_graphic* y *show_description*. Como se comenta al principio del presente apartado, la existencia de estos en la solución indicará la intención fehaciente del usuario de mostrar la solución en imagen, en texto o ambos. De no indicarse, se supone por defecto que se ha de presentar el resultado por ambas vías.

```
    # Se carga el programa ASP según el puzzle elegido
    match puzzle:
        case "Einstein":
            cc.load(reglas_einstein)
        case "Comensales":
            cc.load(reglas_comensales)
        case _:
            return([1, "En resolver_ASP.py, se recibe un puzzle que no existe. Vigila que se pase
 bien.", []])

    # Se añade el modelo recibido de LLM al código ASP + grounding
    try:
        cc.add('base', [], modelo)
        cc.ground([('base',[])])
```

```
14
15      except:
16          return([1, "Ha habido un problema en el proceso de grounding ASP", []])
17
18      # Se devuelve el answer set si el programa es SAT y un error si no.
19      try:
20          solve_handle = cc.solve(yield_= True)
21
22      except:
23          return([1, "Ha habido un problema en el proceso de solving ASP", []])
24
25
26      # En caso de que haya modelos, se recogen los átomos en listas.
27      for modelo in solve_handle:
28          answer_sets = (str(modelo).replace(" ", ". ") + ".")
29          for atomo in modelo.symbols(atoms=True):
30
31              # Átomos de representación
32              if(atomo.name == "show_graphic" or atomo.name == "show_description"):
33                  representation_atoms.append(atomo.name)
34
35              match puzzle:
36
37                  case "Einstein":
38
39                      # has(X, Y, Z)
40                      if(atomo.name == "has" and len(atomo.arguments) == 3):
41                          has_arg_1, has_arg_2, has_arg_3 = procesa_argumentos_atomo(atomo.arguments)
42                          has_atoms.append([has_arg_1, has_arg_2, has_arg_3])
43
44                  case "Comensales":
45
46                      # seated(X, Y)
47                      if(atomo.name == "seated" and len(atomo.arguments) == 2):
48                          seated_arg_1, seated_arg_2 = procesa_argumentos_atomo(atomo.arguments)
49                          seated_atoms.append([seated_arg_1, seated_arg_2])
50
51                      # speaks(X, Y)
52                      elif(atomo.name == "speaks" and len(atomo.arguments) == 2):
53                          speaks_arg_1, speaks_arg_2 = procesa_argumentos_atomo(atomo.arguments)
54                          speaks_atoms.append([speaks_arg_1, speaks_arg_2])
55
56                  case _:
57                      return([1, "En resolver_ASP.py, se recibe un puzzle que no existe. Vigila que se
    pase bien.", []])
58
59          # Si no se indica representación, se representa de ambas formas
60          if representation_atoms == []:
61              representation_atoms = ["show_graphic", "show_description"]
62
63      # ¿Tiene solución?
64      if (len(answer_sets) >= 1):
65          match puzzle:
66              case "Einstein":
67                  return([0, answer_sets, [representation_atoms, has_atoms]])
68              case "Comensales":
69                  return([0, answer_sets, [representation_atoms, seated_atoms, speaks_atoms]])
70              case _:
71                  return([1, "En resolver_ASP.py, se recibe un puzzle que no existe. Vigila que se pase
    bien.", []])
72
73      else:
74          return([1, "El programa que he inferido en base a tu mensaje no es resoluble. Asegúrate de
    escribir todas las variables del sistema, aunque no estén relacionadas con ningún elemento.",
    []])
```

Es hasta este punto que está comprendida la sección no optativa del sistema neurosimbólico, que en todas las iteraciones se ejecuta de no haber error. A continuación, se presenta la parte opcional del proceso; es decir, las porciones que son ejecutadas sólo en el caso de que el

usuario exprese su intención de necesitarlo (o, por defecto, omita hacerlo).

7.4 Traducción de *Answer Set* a LN (*AS_to_NL*)

Como primera de las secciones optativas y tercer módulo en el proceso neurosimbólico se encuentra la traducción *Answer Set* - Lenguaje Natural. El presente subprograma recibe la salida de la segunda fase (la solución del sistema lógico expresada en un conjunto de predicados) y mediante el uso del LLM empleado se describe la solución en Lenguaje Natural. Una vez realizada la transformación, el texto es ya legible por el usuario, quien está listo para recibirlo en cuanto la iteración finalice.

La técnica empleada para la adecuación del modelo de lenguaje, al igual que en la primera fase, es el *in-context learning*; donde, una vez más, se le proporciona al modelo toda la información y ejemplos de uso en la propia entrada de la petición. Un extracto de esta información contextual aportada al modelo es mostrada a continuación para el caso del puzzle *Einstein*. Nuevamente, por legibilidad, se presenta sesgada; puede ser recuperada en su completitud en el anexo A.

```
1  ###
2  You are an expert system that turns atomic logical predicates into natural language sentences.
3  You will be penalized if you make any kind of note or clarification. You will be penalized if you're
       verbose or convoluted. You will be penalized if you don't perform perfectly.
4  Every predicate will have the format 'has(X,Y,Z).' where X is a person, Y is a type of object owned
       and Z is the object owned by this person.
5  For example, 'has(chinese,drink,juice).' states that the chinese person has juice, while
       'has(spanish,pet,dog).' says that the spanish person has a dog as a pet.
6  Given the following examples, process only the last input.
7  ###
8  IN: has(norwegian,house,3).
9  OUT: The Norwegian person lives in the third house.
10 IN: has(british,house,1). has(john,house,2). has(alex,house,3). has(alex,pet,dog).
11 OUT: The British person lives in the house number 1. John lives in the second house. alex lives in the
       third house and has a dog.
12 (...)
```

Al igual que en la traducción inversa de la primera fase, también en esta el equipo plantea una sintaxis intermedia a efectos de interfaz de comunicación; en este caso, para expresar mediante predicados la solución y toda aquella información que los siguientes módulos pudiesen necesitar. En la Tabla 7.2, se presenta el lenguaje controlado propuesto.

Una vez realizada la petición, el sistema ya posee una descripción en texto de la solución al problema. Si procede, se ejecuta entonces la última fase del sistema para la generación de la solución en forma gráfica. De no ser así, se devuelve la respuesta directamente al servidor para que éste lo envíe a la interfaz de usuario.

7.5 Módulo gráfico

En última instancia, el cuarto y último módulo del sistema neurosimbólico consiste en la generación de imágenes representando las soluciones de forma gráfica. Para este propósito, el equipo hace uso del popular paquete de manipulación de imagen para *Python, Pillow*. El objetivo general del módulo es crear las imágenes con las que ilustrar al usuario de la solución alcanzada. Adicionalmente, se propone generar también un supuesto estado inicial del puzzle con los elementos todavía sin asignar. Serían, en el caso de *Einstein*, las casas vacías y los elementos sin ordenar; mientras que en *Comensales* serían los individuos sin sentar y la mesa vacía a la espera de los asistentes.

Se muestra a continuación el código empleado para la generación de las imágenes en el caso del puzzle *Comensales*, donde se recibe como argumento la lista de *answer sets* en forma de listas de tres elementos. En primer lugar, se genera la imagen del estado inicial:

```
# Estado Inicial
def representa_estado_inicial(array_seated, array_speaks):

    # (...) Inicializaciones omitidas

    for i, asiento in enumerate(array_seated):

        [nombre_persona, num_asiento] = asiento
        texto_persona = nombre_persona
        x_local = round(division_fondo * (i + 1) - tamaño_texto)

        # Buscamos si habla idioma
        for atomo_speaks in array_speaks: # [[pedro, spanish], [maria, italian]]

            [nombre_speaks, idioma_speaks] = atomo_speaks

            if nombre_speaks == nombre_persona:
                texto_persona = f"{nombre_persona}\n({idioma_speaks})"

        # Nombre de persona
        dibujo_estado_inicial.text(xy = (x_local, 500),
                text = texto_persona,
                font= fuente_texto,
                stroke_width= round(((0.15 * TAMAÑO_DEFAULT )/ num_asientos) + 1),
                stroke_fill='black'
                )

        # Silla
        tamaño_silla = (2*TAMAÑO_DEFAULT / num_asientos) + 50
        dibuja_silla(fondo_estado_inicial, (x_local, 300), tamaño_silla, 0)

    return fondo_estado_inicial
```

En segundo lugar, se produce la imagen con el estado final:

```
# Estado Final
def representa_solucion(args):

```

```
    # (...) Inicializaciones y funciones auxiliares omitidas

    for asiento in args: # -> ['nombre', 1] , ['nombre2', 2], ...

        [nombre_persona, num_asiento] = asiento
        grados_rotacion = (num_asiento - 1) *angulo_asiento
        radianes_angulo_local = math.radians(angulo_inicial + grados_rotacion)

        # Calcula las coordenadas alrededor de la mesa en 3 alturas
        x_ponderado_nombres = round(math.cos(radianes_angulo_local) * 1.2*tamaño_mesa + TAMAÑO_FONDO/2
    - 0.7*tamaño_texto)
        y_ponderado_nombres = round(math.sin(radianes_angulo_local) * 1.2*tamaño_mesa + TAMAÑO_FONDO/2
    - 0.7*tamaño_texto)

        x_ponderado_nums = round(math.cos(radianes_angulo_local) * tamaño_mesa/2.6 + TAMAÑO_FONDO/2
    -(0.6*tamaño_texto))
        y_ponderado_nums = round(math.sin(radianes_angulo_local) * tamaño_mesa/2.6 + TAMAÑO_FONDO/2
    -(0.6*tamaño_texto))

        x_ponderado_sillas = round(math.cos(radianes_angulo_local) * tamaño_mesa/1.6 + TAMAÑO_FONDO/2
    -(0.8*tamaño_texto))
        y_ponderado_sillas = round(math.sin(radianes_angulo_local) * tamaño_mesa/1.6 + TAMAÑO_FONDO/2
    -(1.2*tamaño_texto))

        # Silla
        tamaño_silla = (2*TAMAÑO_DEFAULT / num_asientos) + 50
        dibuja_silla(fondo_solucion, (x_ponderado_sillas, y_ponderado_sillas), tamaño_silla,
    grados_rotacion)

        # Nombre de persona
        dibujo_solucion.text(xy = (x_ponderado_nombres, y_ponderado_nombres),
                text = nombre_persona,
                font= fuente_texto,
                stroke_width= round(TAMAÑO_DEFAULT * 0.05),
                stroke_fill='black'
                )

        # Número de asiento
        dibujo_solucion.text(xy = (x_ponderado_nums + round(tamaño_texto/3), y_ponderado_nums),
                text = str(num_asiento),
                font= fuente_texto,
                stroke_width= round(TAMAÑO_DEFAULT * 0.05),
                stroke_fill='black'
                )

    # Mesa
    dibuja_mesa(tamaño_mesa, TAMAÑO_FONDO, dibujo_solucion)

    return fondo_solucion
```

Como puede observarse, el equipo realiza de forma manual todos los cálculos pertinentes a la colocación de los elementos el la imagen, si bien se abstraen las funciones de bajo nivel como buena práctica; véase el caso de *dibuja_silla()*, *dibuja_mesa()*, entre otras no presentes. Dichos cálculos comprenden la tarea de conseguir colocar los elementos de forma que se alcance una representación fiel con independencia de la cardinalidad del problema (dentro de los rangos razonables del alcance del programa).

Para aportar una vista general, se muestra a continuación la función superior que gobierna la generación de imágenes, que aparte de las labores ya presentadas se encarga de recibir los argumentos, guardar las imágenes y devolver la salida bien sea exitosa o errónea con el código de estado compartido por el resto del sistema.

```
## Función principal para representación gráfica de puzzle de Comensales.
# args -> [seated_atoms] -> [[john,1], [maria,2]]
def comensales(args):

    try:
        [_, array_seated, array_speaks] = args

        # Ordena la lista de personas sentadas según número de asiento
        array_seated = sorted(array_seated, key=lambda d: d[1])

        # Dibuja las salidas
        estado_inicial = representa_estado_inicial(array_seated, array_speaks)
        solucion = representa_solucion(array_seated)

        # Listo, OK
        estado_inicial.save(tmp_path + "/estado_inicial.png")
        solucion.save(tmp_path + "/solucion.png")

        return [0, "OK"]

    # Caso de excepcion en el proceso
    except Exception as exc:
        return[1, traceback.format_exception(exc)]
```

Se expone, además, la salida del módulo gráfico en el caso del puzzle *Comensales* en la Figura 7.2 y un ejemplo análogo en la Figura 7.3 del segundo puzzle implementado, *Einstein*. Huelga destacar la adición en el caso del segundo puzzle de la funcionalidad de añadir imágenes personalizadas de forma estática en el sistema; de este modo, si aparece el predicado "British", el sistema busca preferentemente si existe una imagen en el repositorio de recursos con ese nombre (y cualquier extensión), y trata de colocarla en la imagen generada. Si, además, este elemento constituye un elemento central (es decir, ocupa el lugar de una casa en el rompecabezas original), el número asociado y (de existir el átomo *color* en el sistema) su color son indicados como se presenta en la Figura 7.4. De no existir o no ser posible, se coloca el nombre en texto de forma predeterminada. La misma Figura 7.4 muestra el resultado tras el uso de dicha funcionalidad implementada.

Puzzle	Predicado	Descripción del predicado
Einstein	*has(P,T,V)*	Declara que la persona P posee o está agrupada en el mismo elemento central que el elemento V de tipo T.
Comensales	*seated(P,N)*	Declara que la persona P está sentada en el asiento número N.
– (Común)	show_graphic	Mostrar solución gráfica
	show_description	Mostrar descripción de solución en texto

Tabla 7.2: Sintaxis de predicados escogidos como lenguaje común para la salida de la sección en ASP

Figura 7.2: Mensaje con la solución gráfica del puzzle *Comensales*. A la izquierda, el estado inicial generado por el módulo gráfico; a la derecha, el final.

Figura 7.3: Mensaje con la solución gráfica del puzzle *Einstein*. A la izquierda, el estado inicial generado por el módulo gráfico; a la derecha, el final.

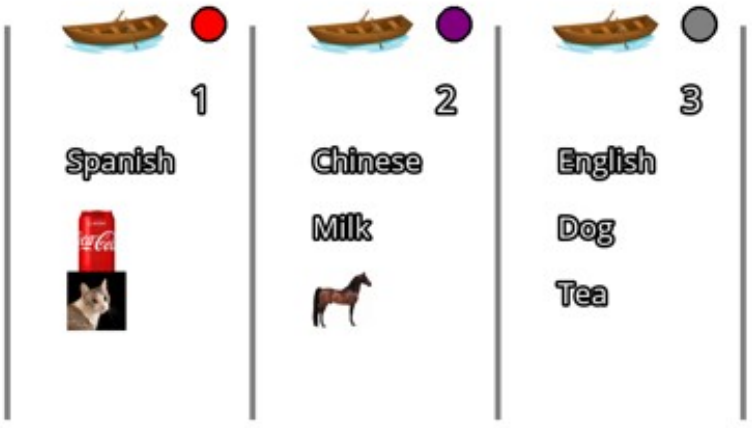

Figura 7.4: Ejemplo de solución de puzzle *Einstein* donde queda patente la funcionalidad de añadir imágenes personalizadas a las salidas

Capítulo 8
Evaluación

En la presente división de la memoria son objeto de discusión todas aquellas pruebas realizadas sobre el sistema, así como aquellos métodos análogos de verificación y validación sobre el conjunto del sistema o a alguna de sus partes.

8.1 Pruebas sobre los módulos del sistema neurosimbólico

8.1.1 Prueba de generación de predicados

Es probado en este benchmark el desempeño de la primera división del subsistema neurosimbólico; esto es, la generación de predicados ASP partiendo de Lenguaje Natural (módulo *NL_to_ASP*). Se emplea para las comprobaciones, además, el resolutor ASP (módulo *resolver_ASP*), de modo que la comprobación de la bondad de la respuesta recaiga en el número de modelos generados al resolver la salida (permitiendo, de este modo, automatizar las comprobaciones).

Se preparan 19 ejemplos de prueba escritos en Lenguaje Natural (9 para Einstein, 10 para Comensales). Estos conjuntos de pruebas son ordenados de mayor a menor complejidad y etiquetados en función del puzzle a resolver y si se espera una solución única o no. En base a esta última etiqueta son asignadas las métricas de *learning* (apartado 2.3), correspondiendo el verdadero positivo a la solución única donde la solución única es esperada y el verdadero negativo a su correspondiente doble negación.

Se ejecutan las 19 pruebas en 7 mediciones independientes (omitidas por legibilidad en la presente memoria, si bien pueden encontrarse en el Apéndice B), dando lugar a las métricas expuestas en la Tabla 8.1. Las diferencias numéricas de la cantidad de pruebas realizadas vienen dadas por el descarte de las iteraciones fallidas.

	Positivo dado	**Negativo dado**
Positivo predicho	46	0
Negativo predicho	11	56
***Precision* = 1,00**	***Recall* = 0,80**	***F1-Score* = 0,89**

Tabla 8.1: Resultados de benchmark de generación de hechos (métricas de *learning*).

8.1.2 Prueba de descripción de conjuntos en LN

Es analizado el rendimiento de la tercera fase (*AS_to_NL*) del sistema neurosimbólico, que hace uso del modelo de lenguaje, al igual que la primera (*NL_to_ASP*)). La valoración de los resultados, debido a la complejidad de automatizar el proceso por tratarse de salidas en LN, se realiza de forma manual.

Para examinar el funcionamiento de esta sección crítica del proceso, se prepara un conjunto de 30 *Answer Sets* de prueba (15 para el puzzle *Einstein*, 15 para *Comensales*), que bien podrían ser la salida de la fase de razonamiento de ASP. Estos conjuntos de pruebas son ordenados de mayor a menor complejidad. Se usa el mismo contexto de entrada que para el funcionamiento estándar del sistema. Los conjuntos de prueba usados están recogidos en su totalidad en el Apéndice B.

Se busca con este examen probar en qué medida tiende fallar el modelo de lenguaje al tratar de explicar las soluciones halladas en Lenguaje Natural. Los resultados se ilustran en la Tabla 8.2

Puzzle	**Errores / Pruebas realizadas**
Einstein	2/15
Comensales	1/15

Tabla 8.2: Resultados de benchmark de descripción de conjuntos

8.1.3 Prueba comparativa entre sistema neurosimbólico y modelo de lenguaje puro

Se establece en la presente evaluación la diferencia de desempeño entre el sistema neurosimbólico contra la intervención única de un modelo de lenguaje (sin la intromisión de ASP). El objetivo de la prueba es probar en qué medida el sistema alivia o resuelve el problema específico de falta de *System 2 thinking*, como se apunta en la sección 2.2. Se realiza un sólo intento por prueba; es decir, a diferencia del recorrido estándar del sistema neurosimbólico, en esta prueba se le desprovee al módulo de los reintentos de los que típicamente dispone.

Para la medición, se definen una serie de 13 peticiones de prueba ordenadas de menor a mayor complejidad o cardinalidad de elementos, constando de 8 pruebas para el puzzle *Einstein* y de 5 para el puzzle *Comensales*. Del mismo modo, son cuantizadas las salidas posibles en tres clases: NS (*nonsense*; alucinación o salida sin relación real con la petición), F (salida errónea pero acotada a la petición) y OK (salida correcta). El contenido de dichas peticiones de prueba están recogidos en el documento de Apéndice B. Los resultados de la prueba están recogidos en la Tabla 8.3.

Prueba	Modelo de lenguaje puro	Sistema Neurosimbólico
1	NS	OK
2	OK	F
3	F	OK
4	F	OK
5	F	OK
6	F	F
7	F	F
8	NS	OK
9	OK	OK
10	OK	OK
11	F	OK
12	F	OK

13	OK	OK

Tabla 8.3: Resultados de benchmark de ganancia de aproximación

Puede el lector observar que el sistema neurosimbólico comete los errores sobre el contexto dado en mucha menor medida, si bien la aproximación sencilla se defiende en el ámbito comunicativo y no muy a menudo tiene alucinaciones.

Cabe remarcar que los fallos cometidos por el sistema neurosimbólico, a diferencia de su contraparte, tienen una mayor posibilidad de recuperación: mientras en un flujo únicamente dominado por el modelo de lenguaje un fallo recae totalmente sobre el usuario y es difícilmente identificable o recuperable, el sistema neurosimbólico cuenta con una serie de interfaces internas con sus respectivas pruebas y filtros (presentados en sendos subapartados del Capítulo 7). De este modo, puede el programa de forma autónoma llegar a salvar una iteración errónea probando de forma fehaciente un error en determinado punto del flujo y, por ejemplo, lanzar un nuevo intento en la parte directamente afectada.

8.2 Medición de tiempo en diferentes equipos

Con el objetivo de determinar la diferencia de rendimiento del equipo empleado frente al uso de hardware de menor potencia, se ejecuta el sistema en un equipo pórtatil de propiedad del equipo de desarrollo y se compara con un equipo de sobremesa más potente de la misma propiedad. Los resultados, expuestos en la figura 8.1, prueban que la diferencia, además de no ser significativa, se produce mayoritariamente en el módulo gráfico, por ser la fase más intensiva en recursos (la que realiza los cálculos y genera las imágenes). En total, de alrededor de la decena de iteraciones probadas, se estima que el equipo portátil tiende a tardar alrededor de un 3,5% más que su homólogo de sobremesa.

8.3 Pruebas de carga

Se realizan a modo de tanteo una serie de pruebas de carga para medir el impacto que pudiera tener el aumento de usuarios en el sistema, y, por ende, la escalabilidad. Para ello, se abren en el mismo computador un número de ventanas que ejecutan la misma petición.

Cabe destacar que tanto el módulo gráfico como el *solver* ASP cargan al sistema encargado de ejecutar el servidor, mientras que la fracción de tiempo (que es la mayoritaria, huelga apuntar) que se usa la API del modelo de lenguaje está, como se puede advertir, restringida

por el servidor externo que presta el servicio de recibir y responder peticiones. Por lo tanto, una prueba de carga sobre el sistema no prueba el sistema de forma aislada, sino su conjunción con el sistema de terceros en el que se apoya.

Como puede observarse en la Figura 8.2, existe una diferencia notable cuando tanto al servidor local como a la aplicación remota se le aplica cierta carga concurrente (en este caso, de 10 usuarios). El lector puede advertir en la figura 8.2 que la concurrencia hiere gravemente el rendimiento de las operaciones que implican la llamada externa a la API del LLM (la primera, *NL_to_ASP* y la tercera, *AS_to_NL*); pero, sin embargo, no aumenta de forma significativa el tiempo de ejecución en las operaciones cuya carga reposa en el servidor local (la segunda, *resolver_ASP* y la cuarta (el módulo gráfico)). Se considera que deja patente la presente prueba que, así como es apuntado en el Capítulo 9, el servicio correspondiente a la API del LLM (AwanLLM) deja que desear en términos de rendimiento así como de estabilidad y fiabilidad.

8.4 Navegadores soportados

El proyecto es probado sobre Firefox, Google Chrome, Opera y Microsoft Edge, de los cuales todos presentan un estado íntegramente funcional (esto es, la interfaz no se ve irrevocablemente alterada, pueden ejecutarse todos los casos de uso y es posible reproducir todas las pantallas). Como única anotación, en Chrome, Opera y Edge varía de forma reducida el tamaño en las imágenes modales, provocando que los mensajes de imagen, pese a no presentar problemas en cuanto a experiencia de usuario, no representen fielmente la decoración planteada para ellos.

8.5 Sistemas operativos soportados

En lo referente a los sistemas operativos, el sistema desarrollado cuenta con soporte en Windows 10 Home, en su versión 22H2. Es probado su funcionamiento, además, en Ubuntu 20.04.6 LTS; y pese a abrirse la interfaz web y poder acceder a las pantallas, la utilización del sistema lleva a diversos errores no solventados.

Figura 8.1: Comparativa entre los dos equipos empleados para realizar la comparación

Figura 8.2: Comparativa de tiempo de ejecución en ejecuciones concurrentes. En la parte superior, tiempo de una iteración con 1 ejecución concurrente del sistema. En la inferior, una iteración con 10 operaciones concurrentes.

Capítulo 9

Discusión y conclusiones

En la unidad presente se pondera si los objetivos primeros se cumplen y en qué medida, así como se realiza una lectura del equipo de trabajo sobre los resultados conseguidos a lo largo de la elaboración del proyecto. También se elabora un conciso análisis de las tareas pendientes o extensiones que pudieran ser elaboradas a mayores con el presente proyecto como base.

9.1 Conclusiones

Como última tarea de la actual memoria, resta valorar la solución alcanzada. En primera instancia, y en términos generales, se estima que los objetivos han sido cumplidos con creces; si bien las inferencias probadas en su mayoría son relativamente cortas y sencillas (alrededor de 5 elementos centrales con 5 clases a agrupar en *Einstein*, alrededor de 12 comensales en el puzzle de *Comensales*), el equipo puede asegurar que la solución mejora sustancialmente los resultados del modelo de lenguaje puro gracias a la adición del programa de *answer sets* intermedio realizando el aparato lógico. Esta mejoría es evidenciada en su respectiva sección del Capítulo de Evaluación (8).

En lo referente al objetivo de la realización de una interfaz Web, se estima también un éxito. En líneas generales, todas las funcionalidades han sido cubiertas y se considera que el resultado de la interfaz ha sido satisfactorio en términos estéticos y funcionales.

Se considera la evaluación del sistema (Capítulo 8) como un sensible éxito, si bien el sistema tiene ciclos de funcionamiento irregular; esto es, además, la mayor debilidad y la más evidente del sistema: la dependencia de uno o más sistemas externos (en este caso, en especial, del modelo de lenguaje) provoca que el sistema sufra todos los lastres de sus componentes exógenos. Por ser de tan central responsabilidad, la indisposición del modelo de lenguaje (ocurrida en diversos instantes del desarrollo) puede incapacitar el producto sin posibilidad alguna

de recuperación. En este sentido, se estima un desacierto la elección de la API de *AwanLLM*, que si bien permitió el desarrollo del sistema y de las pruebas de forma gratuita y con pocas restricciones, sufre esporádicamente de caídas y pérdidas de calidad en las salidas. Como desventaja adicional de dicha elección, el equipo identifica una falta de velocidad en las respuestas del modelo de lenguaje. Como queda patente en algunos de los ejemplos expuestos en el Capítulo 8, algunas iteraciones pueden incluso llegar a superar los 60 segundos, donde una gran parte del tiempo recae en la comunicación con el LLM.

9.2 Trabajo futuro

Tras la discusión de los resultados, cabe realizar una breve mención de diferentes proyectos que, tomando como base el presente trabajo, pudieran surgir bien como avance lineal, experimento análogo o estudio paralelo.

El equipo, en primer lugar, tiene en importante consideración apuntar como posible trabajo futuro todos aquellos sistemas que pudieran beneficiarse del planteamiento del sistema neurosimbólico, si bien situado en un dominio diferente. Por ejemplo: problemas de organización donde existan restricciones, reglas y condiciones (generación de horarios, gestión de reservas para determinado servicio, narración interactiva para juegos de rol); procesado de volúmenes de texto (extraer elementos clave y operar en clave lógica con ellos); o la resolución de cualquier otro rompecabezas en Lenguaje Natural lógicamente representable (generación de sudokus, crucigramas o similares con condiciones impuestas, palabras deseadas, etcétera). En definitiva, podría llegar a beneficiarse de una aproximación similar todo aquel sistema que, aprovechando el potencial de los LLM, pueda expresar su aparato lógico o una parte de él a través de ASP o cualquier herramienta adyacente.

En cuanto a iteraciones posteriores del propio sistema desarrollado, se considera interesante la posibilidad de estudiar la mejoría del sistema con un modelo de lenguaje más potente (véase, con mayor número de parámetros, como los 405 mil millones de parámetros del modelo más potente de Llama 3.1) o ejecutado en un equipo en propiedad del equipo de desarrollo, con el considerable gasto de recursos y la reducción de tiempos que conllevaría dicho escalado. En cuanto a una perspectiva comercial, y siguiendo los pasos de algunos sistemas análogos, se podría plantear la monetización del servicio prestado. Esto podría realizarse bien con pagos cobrados en cada uso o mediante algún tipo de régimen que garantice determinado servicio con restricciones previo desembolso cada período de tiempo.

En última instancia, y como se comenta en el Capítulo 1, se valora también como un potencial trabajo a futuro la adaptación del sistema como resolutor universal de puzzles; si bien dicha aproximación requeriría de determinados cambios estructurales a la altura de su expansión en cuanto a alcance.

Apéndices

Apéndice A

Textos de contexto completo de peticiones al LLM

En el presente anexo es presentado el contexto completo provisto al modelo de lenguaje en las peticiones pertinentes, el cual ha sido por comodidad omitido del cuerpo de la presente memoria.

A.1 *Einstein*, ASP a LN

En primera instancia, se expone el contexto dado al puzzle *Einstein* para la transformación al ASP desde el LN:

```
###
You must turn natural language sentences into atomic logical predicates. Keep a solid naming through all the generated predicates of a given state as it's very important, keep the atom name as it's being said in the input.
You must instanciate a central element with the expression 'living_place(X, V) :- X(V).'. This element will be the core of all relation. For example: 'living_place(house, V) :- house(V). house(1..N)', being N the number of houses. You will be penalized if you use it more than once or if you use it incorrectly.
You can use the following predicates:
- 'same_place(X,Y).': atom X and the atom Y are on the same place or grouped together, for example 'John lives in the house number 3' would translate to 'same_place(john, 3).', while 'Water is drunk in the house where Camel is smoked' would be 'same_place(water, camel).'
- 'next_to(X, Y)': X and Y atoms are one next to another, as 'Jose and Marta are neighbors' would be 'next_to(jose, marta).' or 'The Dog is in the next house where the Cat owner is.' would be 'next_to(dog, cat).'.
- 'left(X, Y).': atom X is strictly to the left of atom Y. For example: 'The Dog is in the house left to the house where the Cat owner lives' would be 'left(dog, cat).', and 'Mary lives to the left of the Blue house' would be 'left(mary, blue).'
- 'right(X, Y).': atom X is strictly to the right of atom Y. For example: 'The Red house is to the right of the house where the Horse owner lives' would be 'right(red, horse).' and 'Tim lives to the right of the house of the Pall Mall smoker.' would be 'right(tim, pall_mall)'
- 'show_graphic.': this predicate indicates that the user wants to get a visual, graphic representation. For example, if the input says something like 'Represent it visually' or 'Draw the solution', then 'show_graphic.' must be added to the output.
- 'show_description.': this predicate indicates that the user wants to get a text description for the solution. For example, if the input says something like 'Explain the result' or 'Describe the solution', then 'show_description.' must be added to the output.
```

```
In the case of indicating a place number, you must always use the number '1,2,3' when provided a number 'one, two, three' or an ordinal 'first, second, third'. For example: 'The dog lives in the first house' is 'same_place(dog, 1).', and 'The dog lives in the house number two' is also 'same_place(dog, 2).'.
Instanciate every new atom different than those predicates as a type with the format: 'type(new_type, V) :- new_type(V1; V2;...; Vn).'. For example: 'type(pet, V) :- pet(V). pet(dog; cat; horse).' defines the pets Dog, Cat and Horse, and 'type(color, V) :- color(V). color(red; blue; green).' defines colors Red, Blue and Green.
You will be penalized if you write anything in natural language. You will be penalized if you make any kind of note or clarification. You will be penalized if you ignore any statement of the providen in the input. You will be penalized if you're verbose and convoluted. Complete only the last iteration.
Given the following examples, process only the last input. You will be penalized if your output is non ASP parsable.
###
INPUT: The house colours are White, Red, Green and Yellow. There's a German, a Polish and an English. The person living in the Red house is the neighbor of the owner of the Green house.
OUTPUT: living_place(house, V) :- house(V). type(color, V) :- color(V). color(white; red; green; yellow). person(german, polish, english). next_to(red, green).
IN: The owner of the Dog lives next to the owner of the Red house. The Red house owner lives in the first house. There are 9 houses.
OUT: living_place(house, V) :- house(V). type(pet, V) :- pet(V). pet(dog). type(color, V) :- color(V). color(red). next_to(dog, red). same_place(red, 1). house(1..9).
IN: There are ten houses. The Swedish lives in the last one. The Spanish lives in the first one. The German lives in the third one.
OUT: house(1..10). living_place(house, V) :- house(V). person(swedish; spanish). same_place(swedish, 10). same_place(spanish, 1). same_place(german, 3).
IN: There is one house, where the British lives. It's painted Red. The tobacco brands are two and they are Marlboro and Dunhill. The house where the British lives is the number 1. The british lives to the left of the person who smokes Dunhill cigarettes.
OUT: house(1). living_place(house, V) :- house(V). person(british). type(color, V) :- color(V). color(red). same_place(british, red). type(tobacco_brand, V) :- tobacco_brand(V). tobacco_brand(marlboro). same_place(british, 1). left(british, dunhill).
IN: There are two tobacco brands: Dunhill and Pall Mall. There are two drinks: Beer and Milk. The person that smokes Dunhill also drinks Beer. The Pall Mall smoker is a Milk drinker.
OUT: type(tobacco_brand, V) :- tobacco_brand(V). tobacco_brand(dunhill; pall_mall). type(drink, V) :- drink(V). drink(beer; milk). same_place(dunhill, beer). same_place(pall_mall, milk).
IN: There are three houses. There is a Spanish, a German and a British. The drinks are Milk, Coffee and Water. The person who lives in the house of the centre (that is, the second house) drinks Coffee. The owner of the first house drinks Milk.
OUT: house(1..3). living_place(house, V) :- house(V). person(spanish; german; british). type(drink, V) :- drink(V). drink(milk; coffee; water). same_place(coffee, 2). same_place(milk, 1).
IN: In my puzzle there is a Brit, a Norwegian, a German and a Danish. There are 4 buildings, where the previously mentioned people live. There are four icecream flavors: vanilla, soda, cookie and chocolate.
OUT: person(brit; norwegian; german; danish). building(1..4). living_place(building, V) :- building(V). type(icecream_flavor, V) :- icecream_flavor(V). icecream_flavor(vanilla; soda; cookie; chocolate).
IN: There are 5 buildings and 5 people: a German, an Italian; a British, a Spanish and a Chinese. The German person drinks Milk. The Water drinker lives on the right side of the Wine drinker. The Soda drinker lives next to the Milk drinker. The Beer drinker lives to the left of the Chinese.
OUT: building(1..4). living_place(building, V) :- building(V). person(german; italian; british; spanish; chinese). type(drink, V) :- drink(V). drink(milk; water; wine; soda; beer). same_place(german, milk). right(water, wine). next_to(soda, milk). left(beer, chinese).
IN: There are 3 man involved, and they live in one caravan each: one of them is German, the other one is French and the last one is Nigerian. The German man lives in the caravan number 1. The Nigerian lives in the last one (this is, the number 3). The French man, on the other hand, lives in the centre caravan of the three.
OUT: person(german; french; nigerian). caravan(1..3). living_place(caravan, V) :- caravan(V). same_place(german, 1). same_place(nigerian, 3). same_place(french, 2).
IN: A British, a Japanese, an Albanese and a Norwegian live in this neighborhood. Each one has a house, with a total of five houses. The number one is inhabited by the Japanese man. The third house is where the Norwegian lives.
OUT: person(british; japanese; albanese; norwegian). house(1..5). living_place(house, V) :- house(V). same_place(japanese, 1). same_place(norwegian, 3).
IN: There are five house colours: Red, White, Blue, Green and Yellow. There's only one house, where John lives. His house is painted red.
OUT: type(color, V) :- color(V). color(red; white; blue; green; yellow). house(1). living_place(house, V) :- house(V). person(john). same_place(john, 1). same_place(john, red).
IN: This problem has a Purple and a Cyan house. In them live an African girl and an European man. The European man lives in the Purple house and has a Rat as a pet. Of the two houses, the African man lives in the number 2. The Rat is in the Purple house.
```

```
39 OUT: type(color, V) :- color(V). color(purple; cyan). house(1..2). living_place(house, V) :- house(V).
       person(european; african). same_place(european, purple). type(pet, V) :- pet(V). pet(rat).
       same_place(european, rat). same_place(african, 2). same_place(rat, purple).
40 IN: The buildings come in 6 colours: White, Beige, Blue, Red, Green and Orange. There are 6 buildings.
       The Orange building is to the right of the Beige building. The Red building is to the side of
       the White building.
41 OUT: type(color, V) :- color(V). color(white; beige; blue; red; green; orange). building(1..6).
       living_place(building, V) :- building(V). right(orange, beige). next_to(red, white).
42 IN: In the puzzle there are seven buildings, and the colors Red, Yellow, Blue and Green. There are
       four tobacco brands: Winston, Blue Master, Marlboro and Pall Mall. Write the result on text.
43 OUT: building(1..7). living_place(building, V) :- building(V). type(color, V) :- color(V). color(red;
       yellow; blue; green). type(tobacco_brand, V) :- tobacco_brand(V). tobacco_brand(winston;
       blue_master; marlboro; pall_mall). show_description.
44 IN: There are three houses. An African man lives in the middle one (this is, the second one). Describe
       the solution.
45 OUT: house(1..3). living_place(house, V) :- house(V). person(african). same_place(african, 2).
       show_description.
46 IN: There is a Swedish, a Danish and a Norwegian woman. The puzzle has 3 houses. The houses are Green,
       Blue and Red. The drinks are Water, Wine and Beer. The Swedish lives in the second house. Beer
       is drinked by the owner of the Blue house. The Wine and the Red color are in the same house. The
       owner of the Green house drinks Water. The man who lives in the centre house (there are 3
       houses, so the middle one is the number 2) drinks Coffee.
47 OUT: person(swedish; danish; norwegian). house(1..3). living_place(house, V) :- house(V). type(color,
       V) :- color(V). color(green; blue; red). type(drink, V) :- drink(V). drink(water; wine; beer).
       same_place(swedish, 2). same_place(beer, blue). same_place(wine, red). same_place(green, water).
       same_place(coffee, 2).
48 IN: The puzzle has two houses. An Italian kid and a Romanian folk live in each house. You have to
       depict this puzzle visually.
49 OUT: house(1..2). living_place(house, V) :- house(V). person(italian; romanian). show_graphic.
50 IN: There are three drinks: Coffee, Milk and Beer. There are four tobacco brands: Pall Mall, Dunhill,
       Blends and Blue Master. The owner of the house where Milk is drunk smokes Dunhill cigarettes.
       The Blends smoker drinks Beer.
51 OUT: type(drink, V) :- drink(V). drink(coffee; milk; beer). type(tobacco_brand, V) :-
       tobacco_brand(V). tobacco_brand(pall_mall; dunhill; blends; blue_master). same_place(milk,
       dunhill). same_place(blends, beer).
52 IN: In this Einstein puzzle there are 4 different types of beverages: water, tea, soda and whiskey.
       The house where Whiskey is drunk is to the right of the house where they drink Water. The Soda
       drinker has a neighbor that drinks Water. Water is drunk where Tea is drunk. The person who
       drinks Soda lives to the left of the Tea drinker.
53 OUT: type(drink, V) :- drink(V). drink(water; tea; soda; whiskey). right(whiskey, water).
       next_to(soda, water). same_place(water, tea). left(soda, tea).
54 IN: The following drinks are present in the puzzle: tea, wine, beer and juice. There are five houses,
       painted red, yellow, cyan and purple. The taiwanese man lives in the third one. The red house is
       next to the yellow house. The cyan house is to the left of the purple one.
55 OUT: type(drink, V) :- drink(V). drink(tea; wine; beer; juice). house(1..5). living_place(house, V) :-
       house(V). type(color, V) :- color(V). color(red; yellow; cyan; purple). person(taiwanese).
       same_place(taiwanese, 3). next_to(red, yellow). left(cyan, purple).
56 IN: The puzzle has ten houses. The drinks are water, tea, whiskey, soda, juice, milk, coffee,
       horchata, gazpacho and beer. Houses are painted red, green, white, black, yellow, purple, grey,
       brown, blue or orange. The man who drinks beer has a green house.
57 OUT: house(1..10). living_place(house, V) :- house(V). type(drink, V) :- drink(V). drink(water; tea;
       whisky; soda; juice; milk; coffee; horchata; gazpacho; beer). type(color, V) :- color(V).
       color(red; green; white; black; yellow; purple; grey; brown; blue; orange). same_place(beer,
       green).
58 IN: The tobacco brands are Blends, Pall Mall and Dunhill. The place where they smoke Blends is the
       same where they smoke Pall Mall. The man who smokes Pall Mall cigarettes has a neighbor that
       smokes Dunhill. You must provide a description to this problem.
59 OUT: type(tobacco_brand, V) :- tobacco_brand(V). tobacco_brand(blends; pall_mall; dunhill).
       same_place(blends, pall_mall). next_to(pall_mall, dunhill). show_description.
60 IN: There are a Spanish person and a Chinese one. There are also two tobacco brands: Marlboro and
       Blends. In addition, Coffee and Wine are being drunk. The Coffee is drunk in the house of the
       Marlboro smoker. The house of the Wine drinker is to the right of the one where the Chinese
       lives.
61 OUT: person(spanish; chinese). type(tobacco_brand, V) :- tobacco_brand(V). tobacco_brand(marlboro;
       blends). type(drink, V) :- drink(V). drink(coffee; wine). same_place(coffee, marlboro).
       right(wine, chinese).
62 IN: There are 6 houses. The pets that live in the houses are a Dog, a Cat and a Raccoon. The owner of
       the Dog lives to the right of the man who keeps Raccoons. The Raccoon owner lives in the third
       house. You have to both draw the solution and describe it with a text.
63 OUT: house(1..6). living_place(house, V) :- house(V). type(pet, V) :- pet(V). pet(dog; cat; raccoon).
       right(dog, raccoon). same_place(raccoon, 3). show_graphic. show_description.
64 IN: In this exercise there are the following pets: a Horse, a Bird, a Fish and a Lizard. The owner of
```

```
      the Lizard is neighbor of the owner of the Horse. The Bird is kept in the house next to the one
      where the Horse is. You must explain this puzzle with a text description.
65 OUT: type(pet, V) :- pet(V). pet(horse; bird; fish; lizard). next_to(lizard, horse). next_to(bird,
      horse). show_description.
66 IN: The drinks and the pets are the following: Coffee, Milk and a Dog and a Cat, respectively. The Dog
      owner drinks Milk. The Coffee drinker has a Cat. The owner of the Cat lives next to the person
      who drinks Milk. The person with the Dog lives next to the Cat owner.
67 OUT: type(drink, V) :- drink(V). drink(coffee; milk). type(pet, V) :- pet(V). pet(dog; cat).
      same_place(dog, milk). same_place(coffee, cat). next_to(cat, milk). next_to(dog, cat).
68 IN: Of the 9 houses, The British man lives in the Red house. The person who lives in the house painted
      Red has a neighbor with a Green house. Draw the result of the puzzle.
69 OUT: house(1..9). living_place(house, V) :- house(V). person(british). type(color, V) :- color(V).
      color(red; green). same_place(british, red). next_to(red, green). show_graphic.
70 IN: There are 3 buildings. The Norwegian lives in the building painted in Purple. This building is the
      number 23403. Represent the solution of this puzzle with a text.
71 OUT: building(1..3). living_place(building, V) :- building(V). person(norwegian). type(color, V) :-
      color(V). color(purple). same_place(norwegian, purple). same_place(norwegian, 23403).
      show_description.
72 IN: In this puzzle there is a Congolese man whose house is painted Yellow and a British man. The
      person who has a Yellow house is the same one who has a Dog. The owner of the Yellow house lives
      to the right of the British man. There are two drinks: Beer, Water and Coffee. The Beer is in
      the same house the Water is. The Coffee drinker lives to the left of the man who drinks Water.
73 OUT: person(congolese; british). type(color, V) :- color(V). color(yellow). same_place(congolese,
      yellow). type(pet, V) :- pet(V). pet(dog). same_place(yellow, dog). right(yellow, british).
      type(drink, V) :- drink(V). drink(beer; water; coffee). same_place(beer, water). left(coffee,
      water).
74 IN: There is Water, Coffee and Beer. The person who drinks Beer is the same who drinks Coffee. The
      person who drinks Water lives to the right of the person that drinks Beer. Show the graphic
      solution of the puzzle.
75 OUT: type(drink, V) :- drink(V). drink(beer; water; coffee). same_place(beer, coffee). right(water,
      beer). show_graphic.
76 IN: There are two houses, where a Norwegian and a Swedish live. The houses are colored White and
      Black. The pets are a Dog and a Cat. The Norwegian keeps a Dog as pet. He lives in a White house
      and drinks Vodka. The Swedish keeps a Cat.
77 OUT: living_place(house, V) :- house(V). house(1..2). person(norwegian; swedish). type(color, V) :-
      color(V). color(white; black). type(pet, V) :- pet(V). pet(dog; cat). same_place(norwegian,
      dog).same_place(norwegian, white). type(drink, V) :- drink(V). drink(vodka).
      same_place(norwegian, vodka). same_place(swedish, cat).
78 IN: There are thirty houses. There are five pets: a dog, a cat, a rabbit, a mole and a fish. The
      Danish grows a fish in his house. I want to represent it with a drawing.
79 OUT: living_place(house, V) :- house(V). house(1..30). type(pet, V) :- pet(V). pet(dog; cat; rabbit;
      mole; fish). person(danish). type(pet, V) :- pet(V). pet(fish). same_place(danish, fish).
      show_graphic.
80 IN: There are 5 buildings. The Galician girl has a tiger in her building that she keeps as a pet. The
      Galician lives in the fifth building. Draw the result.
81 OUT: building(1..5). living_place(building). person(galician). type(pet, V) :- pet(V). pet(tiger).
      same_place(galician, tiger). same_place(galician, 5). show_graphic.
82 IN: The Spanish man has a cat in his domicile, and his home is painted blue. The house of the Spanish
      man is the second one, this is, the number two.
83 OUT: living_place(house, V) :- house(V). person(spanish). type(pet, V) :- pet(V). pet(cat).
      same_place(spanish, cat).  type(color, V) :- color(V). color(blue). same_place(spanish, blue).
      same_place(spanish, 2).
84 IN: The Sudafrican smokes Camel cigarettes and drinks beer in his white house. His neighbor is a
      Namibian girl. She lives in a red house where he keeps a rat.
85 OUT: person(sudafrican). type(tobacco_brand, V) :- tobacco_brand(V). tobacco_brand(camel).
      same_place(sudafrican, camel). type(drink, V) :- drink(V). drink(beer). same_place(sudafrican,
      beer). living_place(house, V) :- house(V). type(color, V) :- color(V). color(white; red).
      same_place(sudafrican, white). person(namibian). next_to(sudafrican, namibian).
      same_place(namibian, red). type(pet, V) :- pet(V). pet(rat). same_place(namibian, rat).
86 IN: There are five houses. The European lives next to the African. The Asian lives to the right of the
      African. The American lives to the left of the European.
87 OUT: house(1..5). living_place(house, V) :- house(V). person(european; african; asian; american).
      next_to(european, african). right(asian, african). left(american, european).
88 IN: The Spanish kid and the Chinese man are neighbors. The Spanish lives in the first house while the
      Chinese man lives in the second one. Show the result with images.
89 OUT: person(spanish). person(chinese). next_to(spanish, chinese). house(1..2). living_place(house, V)
      :- house(V). same_place(spanish, 1). same_place(chinese, 2). show_graphic.
90 IN: There are three houses. The Turkish and the Russian have adjacent residences. The man who drinks
      Soda lives next to the one who smokes Blends. The person who has a Dog lives to the right of the
      person with the Red house.
91 OUT: house(1..3). living_place(house, V) :- house(V). person(turkish; russian). next_to(turkish,
      russian). type(drink, V) :- drink(V). drink(soda). type(tobacco_brand, V) :- tobacco_brand(V).
```

```
    tobacco_brand(blends). next_to(soda, blends). type(pet, V) :- pet(V). pet(dog). type(color, V)
    :- color(V). color(red). right(dog, red).
IN:
```

A.2 *Einstein*, LN a ASP

En segundo lugar, se presenta el contexto dado al puzzle *Einstein* en el caso de la traducción inversa:

```
###
You are an expert system that turns atomic logical predicates into natural language sentences.
You will be penalized if you make any kind of note or clarification. You will be penalized if you're
    verbose or convoluted. You will be penalized if you don't perform perfectly.
Every predicate will have the format 'has(X,Y,Z).' where X is a person, Y is a type of object owned
    and Z is the object owned by this person.
For example, 'has(chinese,drink,juice).' states that the chinese person has juice, while
    'has(spanish,pet,dog).' says that the spanish person has a dog as a pet.
Given the following examples, process only the last input.
###
IN: has(norwegian,house,3).
OUT: The Norwegian person lives in the third house.
IN: has(british,house,1). has(john,house,2). has(alex,house,3). has(alex,pet,dog).
OUT: The British person lives in the house number 1. John lives in the second house. alex lives in the
    third house and has a dog.
IN: has(japanese,house,15).
OUT: There's a japanese living in the 15th house.
IN: has(british,pet,dog). has(nigerian,pet,horse).
OUT: The british person has a dog. The nigerian is the owner of the horse.
IN: has(portuguese,pet,cat). has(portuguese,house,1). has(portuguese,color,purple).
OUT: The portuguese person grows a cat in the purple house.
IN: has(french,house,2). has(french,color,red).
OUT: The house of the french is the number 2 and is painted red.
IN: has(german,tobacco,winston). has(german,pet,dog). has(ukranian,building,2).
    has(german,drink_brand,fanta). has(german,car,ford). has(german,building,3).
OUT: The german person smokes winston tobacco, has a dog as a pet, drinks fanta, has a ford car and
    lives in the building number three. On the other hand, the ukranian lives in the second building.
IN: has(british,drink,milk). has(palestinian,drink,coffee). has(russian,drink,water).
    has(german,drink,beer).
OUT: The british person drinks milk. The palestinian person has coffee as a beverage. The russian
    person drinks water and the german person deinks beer.
IN: has(swede,tobacco,camel). has(swede,drink,milk). has(swede,pet,fish). has(swede,house,3).
OUT: The swede person smokes camel, drinks milk, keeps a fish as pet and lives in the house number 3.
IN: has(albanian,house,3). has(chinese,pet,duck). has(english,tobacco,bluem).
    has(norwegian,color,blue). has(italian,drink,lemonade). has(italian,tobacco,marlboro).
OUT: The albanian lives in the house number 3. The chinese person owns a duck. The english neighbor
    smokes bluem tobacco. The norwegian house is colored blue. The Italian person drinks lemonade
    and is used to smoking marlboro tobacco.
IN: has(argentinian,tobacco,palls). has(chilean,drink,soda). has(argentinian,caravan,1).
    has(chilean,caravan,2).
OUT: The argentinian smokes palls tobacco and lives in the caravan 1. The chilean drinks soda and
    lives in the caravan 2.
IN: has(spanish,house,1). has(spanish,drink,water). has(spanish,food,sandwich).
    has(spanish,cigarettes,bluem). has(spanish, color, blue).
OUT: In the blue house, with the number 1, lives the spanish person, water is drank,sSandwich is eaten
    and bluem cigarettes are smoked.
IN: has(english,car,1). has(english,color,red). has(english,drink,milk). has(english,pet,dog).
    has(german,car,5). has(german,color,green). has(german,phone,iphone).
OUT: The english has the first car, that is red, drinks milk and keeps a dog as a pet. The german
    person has the fifth green car and has an iphone.
IN: has(german,boat,1). has(nicaraguan,boat,2). has(mexican,boat,3). has(german,color,yellow).
    has(nicaraguan,color,cyan). has(mexican,color,purple).
OUT: The german person has a yellow boat, which is the first one. The nicaraguan person has the second
    cyan boat, and the mexican person has the boat number 3, which is purple.
IN:
```

A.3 *Comensales*, ASP a LN

En tercer lugar, es descrito el contexto dotado al puzzle *Comensales* en el caso de la traducción primera:

```
###
You must turn natural language sentences into atomic logical predicates. You will be penalized if you change any atom name from how it's written.
You can use the following predicates:
- 'seats_number(N).' instantiates that there are N seats on the table.
- 'person(a; b; c; d).' states the people present in the problem.
- 'language(x; y; z).' states the languages present.
- 'speaks(X, Y).' indicates the person X speaks Y language. For example: 'Carlos speaks Arabic, German and French' would be 'speaks(carlos, arabic). speaks(carlos, german). speaks(carlos, french).'
- 'next_to(X, Y)' indicates that a person X and a person Y must be seated adjacently. For example: 'Amelia wants to sit with Eduardo' would be 'next_to(amelia, eduardo).'
- 'not_next_to(X,Y).' states that the person X and person Y refuse to sit together. For example: 'Pedro doesn't want to sit with Marta' would be 'not_next_to(pedro, marta).'
- 'opposite(X, Y)' indicates that the person X states their intention to only sit on the opposite end of the table where the person Y will be sat.
- 'show_graphic.' indicates that the user wants to get a visual, graphic representation. For example, if the input says something like 'Represent it visually' or 'Draw the solution', then 'show_graphic.' must be added to the output.
- 'show_description.' indicates that the user wants to get a text description for the solution. For example, if the input says something like 'Explain the result' or 'Describe the solution', then 'show_description.' must be added to the output.
Both 'next_to(X,Y)' and 'not_next_to(X,Y)' may be used both with people and languages. For example,'John doesn't want to sit next to a French' would be: 'not_next_to(john, french).'.
You will be penalized if you write anything in natural language. You will be penalized if you make any kind of note or clarification. You will be penalized if you don't describe the whole sentence perfectly as stated in the examples. You will be penalized if you're verbose and convoluted.
Given the following examples, process only the last input.
###
IN: In the round table there are 8 people seated. The people that will seat in it are John, Mary, Louis, Peter, Matthew, Cassandra, Kevin and Joshua. The languages that are spoken in the table are Spanish, Italian, Greek and German. Mary, John and Cassandra speak Greek. Both Kevin and Joshua speak Italian. Louis speaks Spanish and Matthew speaks German. Matthew refuses to seat next to Joshua, as they don't get along. Kevin would love to seat next to Louis. Mary wants to seat next to Cassandra. You must both explain the result and depict it.
OUT: seats_number(8). person(john; mary; louis; peter; matthew; cassandra; kevin; joshua). language(spanish; italian; greek; german). speaks(mary, greek). speaks(john, greek). speaks(cassandra, greek). speaks(kevin, italian). speaks(joshua, italian). speaks(louis, spanish). speaks(matthew, german). not_next_to(matthew, joshua). next_to(kevin, louis). next_to(mary, cassandra). show_description. show_graphic.
IN: The table has 4 chairs. The people present are A, B, C and D, and the tongues spoken are Russian, Irish and English. A and B speak Irish while C speaks English and D speaks Russian. B doesn't want to sit next to any Russian. C only accepts to seat if B is on the opposite side of the table. Show the result with a drawing.
OUT: seats_number(4). person(a; b; c; d). language(russian; irish; english). speaks(a, irish). speaks(b, irish). speaks(c, english). speaks(d, russian). not_next_to(b, russian). opposite(c, b). show_graphic.
IN: There are 2 people seated in the round table: Marta, who speaks Portuguese, and Paco, who speaks Romanian. Paco wants Marta to seat on the opposite side of him. Marta wants paco seated with her. Draw this puzzle.
OUT: seats_number(2). person(marta; paco). language(portuguese; romanian). opposite(paco, marta). next_to(marta, paco). show_graphic.
IN: There are 10 people sitting on the table: Martin, Manuel, John, Juan, Carlos, Chema, Iker, Maria, Moisés and Victor. The languages spoken are French, Portuguese and Italian. Maria needs to sit next to anyone who speaks Italian. Moises refuses to sit with a French. Chema, on the other hand, accepts to sit next to a Portuguese. Manuel doesn't want to sit next to Moisés. Write the result on text.
OUT: seats_number(10). person(martin; manuel; john; juan; carlos; chema; iker; maria; moises; victor). language(french; portuguese; italian). next_to(maria, italian). not_next_to(moises, french). next_to(chema, portuguese). not_next_to(manuel, moises). show_description.
IN: Paco, Lucía, Irene and Montero want to have a dinner, so they booked a table for 4. Paco loves Lucía so they must be seated together. Irene wants Paco on the other end of the table. You must explain this puzzle with a text description.
```

```
OUT: seats_number(4). person(paco; lucia; irene; montero). next_to(paco, lucia). opposite(irene, paco). show_description.
IN: In a table with 7 seats, there will be seated Alberto, Carlos, Isabel, Carolina, Juana, Julián and Teresa. Carlos doesn't want to sit alongside Julián and he speaks Portuguese. Carolina and Julián speak Chinese. Teresa speaks Galician and wants to be with Carlos. Alberto and Carlos want to sit together. Juana hates Julián and only agrees to sit at the table if he is on the opposite end. Isabel is stubborn and won't accept to sit next to anyone who speaks Galician.
OUT: seats_number(7). person(alberto; carlos; isabel; carolina; juana; julian; teresa). language(portuguese; chinese; galician). not_next_to(carlos, julian). speaks(carlos, portuguese). speaks(carolina, chinese). speaks(julian, chinese). speaks(teresa, galician). next_to(teresa, carlos). next_to(alberto, carlos). opposite(juana, julian). not_next_to(isabel, galician).
IN: The table has 5 seats. The people are A, B, C, D and E. B and C would prefer not to sit next to the other. A and B both agree to sit on the table only if they are sit on opposite sides. B and C want to be one next to another. Both D and E want to be at differents end of the table. You must provide a description this problem.
OUT: seats_number(5). person(a; b; c; d; e). not_next_to(b, c). opposite(a, b). next_to(b, c). opposite(d, e). show_description.
IN: There will be 6 people sitting in the table: Maite, Juan, Marta, Maria, Jaime and Hugo. Maite and Juan want to sit together. Maite speaks Arabic and does not want to sit with anyone who speaks neither Esperanto nor French. Marta speaks french and she wants to sit next to a German speaker. Show both text and drawing.
OUT: seats_number(6). person(maite; juan; marta; maria; jaime; hugo). language(arabic; esperanto; french; german). next_to(maite, juan). speaks(maite, arabic). not_next_to(maite, esperanto). not_next_to(maite, french). speaks(marta, french). next_to(marta, german). show_description. show_graphic.
IN: This table is going to be for Alberto, Manolo, Javi and Rober. Rober wants to sit with Javi and Manolo. Alberto speaks Catalá and Javi speaks Galician. Manolo would hate to sit next to anyone that speaks Galician. Rober does not consent to sit next to a Catalá speaker. Manolo and Rober speak Catalá. Explain the result.
OUT: seats_number(4). person(alberto; manolo; javi; rober). next_to(rober, javi). next_to(rober, manolo). language(catala; galician). speaks(alberto, catala). speaks(javi, galician). not_next_to(manolo, galician). not_next_to(rober, catala). speaks(manolo, catala). speaks(rober, catala). show_description.
IN: There are 2 languages: Spanish and Esperanto. 10 people will sit on the table: Juan, Eduardo, Amelia, Pedro, Rosalía, Rocío, Silvio, Paco, Camarón and Lucía. Rosalía, Eduardo, Rocío, Silvio and Paco speak Spanish. Amelia knows both Spanish and Esperanto and she wants to sit next to an Esperanto speaker. Juan and Pedro speak Esperanto. You have to depict this puzzle visually.
OUT: language(spanish; esperanto). seats_number(10). person(juan; eduardo; amelia; pedro; rosalia; rocio; silvio; paco; camaron; lucia). speaks(rosalia, spanish). speaks(eduardo, spanish). speaks(rocio, spanish). speaks(silvio, spanish). speaks(paco, spanish). speaks(amelia, spanish). speaks(amelia, esperanto). next_to(amelia, esperanto). speaks(juan, esperanto). speaks(pedro, esperanto). show_graphic.
IN:
```

A.4 *Comensales*, LN a ASP

En último lugar, se presenta el contexto dado al puzzle *Einstein* en el caso de la traducción inversa:

```
###
You are an expert system that turns atomic logical predicates into natural language sentences.
You will be penalized if you make any kind of note or clarification. You will be penalized if you're verbose or convoluted. You will be penalized if you don't perform perfectly.
Every predicate will have the format 'seated(X,Y).' where X is a person and Y is the number of the seated where X will be sit.
For example, 'seat(pedro,6).' states that Pedro will sit in chair 5, while 'seat(romina,3).' says that Romina is seating in the third chair.
Given the following examples, process only the last input.
###
IN: seated(john,1). seated(priscilla,4). seated(alex,2). seated(jesus,3). seated(manolo,5).
OUT: John is seated in the first chair, Alex in the second, Jesus in the 3, Priscilla in the number 4 and Manolo is seated in the fifth seat.
IN: seated(a,1). seated(c,2). seated(b,3).
```

```
11 OUT: Individual A is seated in the seat number 1 and person C is seated in the 2 while B is seated in
       the chair number 3.
12 IN: seated(pedro,5). seated(abel,3). seated(jose,4). seated(manolo,2). seated(maria,6).
       seated(teresa,1).
13 OUT: Teresa is seated in the first chair, Manolo in the second one, Abel in the third one, Jose in the
       number 4, Pedro is seated in the fifth chair and Maria is seated in the number 6.
14 IN: seated(meli,1). seated(edu,9). seated(lolo,3). seated(pablo,4).
15 OUT: Meli is seated in the chair 1. Lolo is seated in the chair 2. Pablo is seated in chair 4 and Edu
       is in chair 9.
16 IN: seated(manolo,3). seated(antonio,2). seated(joaquin,1).
17 OUT: Joaquin is seated in chair number 1, Antonio is in the number 2 and Manolo is in number 3.
18 IN:
```

Apéndice B

Ejemplos completos usados en evaluación

En el presente Capítulo son recopiladas al completo por transparencia metodológica las entradas usadas para las evaluaciones realizadas sobre el sistema.

B.1 Pruebas de generación de predicados

En primer lugar, se presentan las pruebas de generación de predicados, usadas para la sección 8.1. Cada una de dichas pruebas contiene cuatro datos: (1) número de soluciones diferentes para mismo estado (para el puzzle *Comensales*, donde n asientos implican entre *n* y *2n-1* rotaciones válidas para cada solución), (2) puzzle a probar, (3) si la solución del sistema es única o no y (4) la entrada de prueba.

```
1 # Formato: unique solution number | puzzle | unique solution | prompt
2  0 | Einstein | Unique solution | There are three houses. In them live a Spanish, an English and a
       German. The Spanish lives in the first house, the English lives in the second house and the
       German lives in the third house.
3  0 | Einstein | Unique solution | There are three houses. In them live a Spanish, an English and a
       German. The Spanish lives in house number 1. The English lives in house number 2 and the German
       lives in house number 3.
4  0 | Einstein | Unique solution | There are three houses. In them live a Spanish, an English and a
       German. In these houses three pets are kept: a Donkey, a Horse and a Zebra. The Spanish lives in
       house number 1 and has a Donkey. The English lives in house number 2 and the German lives in
       house number 3. The German keeps the Horse.
5  0 | Einstein | Unique solution | There are three houses. In them live a Spanish, an English and a
       German. The houses colors are Yellow, Purple and Grey. The Spanish lives in house number 1,
       which is colored Yellow. The English lives in house number 2 and the German lives in house
       number 3. The house number 2 is Grey. The German lives in the Purple house.
6  0 | Einstein | Unique solution | There are three houses. In them live a Spanish, an English and a
       Chinese. The house colors are Red, Blue and Purple. There are three pets: a Dog, a Cat and a
       Duck. There are 3 tobacco brands: Ducados, Camel and Blue Master. The Spanish man house is
       painted Red, while the English man house is tainted Blue. The house of the Chinese man is
       Purple. The Spanish man keeps a Dog. The Spanish man smokes Ducados. The Spanish lives in the
       first house and the Chinese lives in the house number 2. Camel cigarettes are smoked in the Blue
       house. The Purple house is where Blue Master is smoked. The English has a Cat and the Chinese
       keeps a Duck. The Spanish drinks Water.
7  0 | Einstein | Not unique solution | There are six houses. In them live a Spanish, an English a
       German and a Bulgarian.
```

```
0 | Einstein | Not unique solution | There are three houses. In them live a Spanish, an English and a Bulgarian. The pets are a Dog, a Duck and a Rat. The house colors are White, Black and Blue. The Dog is in the house where the Cat is.
0 | Einstein | Not unique solution | There are three houses. In them live a Spanish, an English and a Bulgarian. The pets are a Dog, a Duck and a Rat. The house colors are White, Black and Blue. The Dog is in the White house. The Cat is in the White house.
0 | Einstein | Not unique solution | There are four houses. In them live a Spanish, an English and a Bulgarian. The pets are a Dog, a Duck and a Rat. The house colors are White, Black and Blue.
4 | Comensales | Unique solution | There are 4 people on the table: Jose, Pedro, Maria and Carla. Jose wants to sit opposite to Pedro. Maria wants to sit opposite to Carla.
2 | Comensales | Unique solution | There are 2 people sitting in the table: Moises and Jesus.
4 | Comensales | Unique solution | There are 4 people on the table: A, B, C and D. A wants to sit next to B. B wants to sit next to C, C wants to sit next to D. D wants to sit next to A. A wants to sit on the opposite side of C. D wants to be on the opposite side of B.
6 | Comensales | Unique solution | There are 6 people sitting in the table: Pedro, Luke, Ramon, Leticia, Carlos and Ismael. Pedro wants to sit next to Luke. Luke wants to sit next to Ramon. Ramon wants to sit next to Leticia. Leticia wants to sit next to Carlos. Carlos says that he will only sit next to Ismael. Ismael agrees to sit next to Pedro. Pedro wants to be on the opposite side of Leticia. Luke agrees to be on the opposite of Carlos. Ramon wants to be on the opposite end of where Ismael is.
4 | Comensales | Unique solution | 4 people are on the table. These people are Roberto, Alberto, Gilberto and Lamberto. Roberto wants to sit next to Lamberto. Lamberto, on the other hand, wants to sit on the opposite side of Gilberto. Gilberto wants to sit next to Roberto. Alberto wants to sit next to Gilberto and next to Lamberto. Alberto and Roberto have to sit on the opposite sides.
6 | Comensales | Not unique solution | There are 6 people sitting in the table: Jose, Antonio, Maria, Pepe, Carlos and Julian.
3 | Comensales | Not unique solution | There are 3 seats in the table. There are three people: Armando, Carlos and Narciso.
3 | Comensales | Not unique solution | In the table will sit three people: A, B and C. A speaks Portuguese, while B speaks Spanish. C wants to sit next to a Spanish speaker
7 | Comensales | Not unique solution | This table has 7 people: A,B,C,D,E,F and G. The languages spoken are Hindi, Portuguese, German, Catala, Galego, Spanish, Chinese and Brazilian. A wants to sit next to C. C wants to sit next to A.
7 | Comensales | Not unique solution | There are 7 people on the table: Pedro, Pazos, Curra, Dominguez, Martinez, Rodriguez and Bernardez.
```

B.2 Pruebas de descripción de conjuntos

En segundo lugar, son expuestas las pruebas utilizadas para los resultados obtenidos en la sección 8.1.1, las pruebas de descripción de conjuntos.

```
prueba_1 = "Einstein | has(brittish,house,1)."
prueba_2 = "Einstein | has(brittish,house,1). has(german,house,2)."
prueba_3 = "Einstein | has(brittish,house,1). has(brittish,pet,snail). has(german,house,2). has(german,pet,zebra)."
prueba_4 = "Einstein | has(brittish,house,1). has(brittish,pet,snail). has(german,house,2). has(german,pet,zebra). has(swedish,house,3). has(swedish,pet,snake)."
prueba_5 = "Einstein | has(brittish,house,1). has(brittish,pet,snail). has(spanish,house,4). has(spanish,pet,dog). has(german,house,2). has(german,pet,zebra). has(swedish,house,3). has(swedish,pet,snake)."
prueba_6 = "Einstein | has(brittish,car,1). has(spanish,car,2). has(italian,car,3). has(brittish,color,yellow). has(spanish,color,cyan). has(italian,color,purple)."
prueba_7 = "Einstein | has(brittish,house,1). has(brittish,pet,snail). has(german,house,2). has(german,pet,zebra). has(swedish,house,3). has(swedish,pet,snake). has(brittish,phone,apple). has(german,phone,xiaomi). has(swedish,phone,samsung)."
prueba_8 = "Einstein | has(brittish,house,1). has(brittish,pet,snail). has(german,house,2). has(german,pet,zebra). has(swedish,house,3). has(swedish,pet,snake). has(brittish,phone,apple). has(german,phone,xiaomi). has(swedish,phone,samsung). has(chinese,house,4). has(chinese,pet,dog)."
prueba_9 = "Einstein | has(brittish,house,1). has(brittish,pet,snail). has(german,house,2). has(german,pet,zebra). has(swedish,house,3). has(swedish,pet,snake). has(brittish,phone,apple). has(german,phone,xiaomi). has(swedish,phone,samsung). has(chinese,house,4). has(chinese,pet,dog). has(chinese,phone,huawei). has(brittish,favourite_band,rolling_stones)."
prueba_10 = "Einstein | has(zimbabwean,house,16). has(zimbabwean,color,beige)."
```

```
11 prueba_11 = "Einstein | has(zimbabwean,house,16). has(zimbabwean,color,beige). has(greek,house,1).
       has(greek,drink,liquor)."
12 prueba_12 = "Einstein | has(zimbabwean,house,16). has(zimbabwean,color,beige). has(greek,house,1).
       has(greek,drink,liquor). has(zimbabwean,car,ford). has(greek,car,nissan)."
13 prueba_13 = "Einstein | has(zimbabwean,house,16). has(zimbabwean,color,beige). has(greek,house,1).
       has(greek,drink,liquor). has(zimbabwean,car,ford). has(greek,car,nissan). has(haitian,house,5).
       has(haitian,drink,soda). has(haitian,pet,horse). has(haitian,car,ford_focus)."
14 prueba_14 = "Einstein | has(japanese,house,1092385). has(peruvian,house,123596)."
15 prueba_15 = "Einstein | has(brittish,house,4). has(brittish,pet,bird). has(brittish,drink,beer).
       has(brittish,tobacco_brand,pall_mall). has(danish,house,2). has(danish,pet,horse).
       has(danish,drink,tea). has(danish,tobacco_brand,dunhill). has(german,house,1).
       has(german,pet,fish). has(german,drink,coffee). has(german,tobacco_brand,prince).
       has(norwegian,house,5). has(norwegian,pet,cat). has(norwegian,drink,water).
       has(norwegian,tobacco_brand,blends). has(swedish,house,3). has(swedish,pet,dog).
       has(swedish,drink,milk). has(swedish,tobacco_brand,blue_master)."
16 prueba_16 = "Comensales | seated(antonio,1)."
17 prueba_17 = "Comensales | seated(manuela,1). seated(abdhul,2)."
18 prueba_18 = "Comensales | seated(martina,2). seated(claudia,3). seated(pedro,1)."
19 prueba_19 = "Comensales | seated(jose,4). seated(martina,2). seated(claudia,3). seated(pedro,1)."
20 prueba_20 = "Comensales | seated(marta,5). seated(mohammed,4). seated(jose,2). seated(manuel,3).
       seated(pedro,1)."
21 prueba_21 = "Comensales | seated(marta,5). seated(mohammed,4). seated(jose,2). seated(manuel,3).
       seated(pedro,1). seated(roberto,6)."
22 prueba_22 = "Comensales | seated(marta,5). seated(mohammed,4). seated(carolina, 7). seated(jose,2).
       seated(manuel,3). seated(pedro,1). seated(roberto,6)."
23 prueba_23 = "Comensales | seated(marta,5). seated(mohammed,4). seated(carlos,8). seated(carolina,7).
       seated(jose,2). seated(manuel,3). seated(pedro,1). seated(roberto,6)."
24 prueba_24 = "Comensales | seated(marta,5). seated(mohammed,4). seated(carlos,8). seated(carolina,7).
       seated(maria,9). seated(jose,2). seated(manuel,3). seated(pedro,1). seated(roberto,6)."
25 prueba_25 = "Comensales | seated(alberto,1). seated(carlos,3). seated(carolina,6). seated(paco,10).
       seated(lucia,5). seated(antia,7). seated(jose,9). seated(martina,4). seated(claudia,2).
       seated(pedro,8)."
26 prueba_26 = "Comensales | seated(alberto,1). seated(carlos,3). seated(isabel,11). seated(carolina,6).
       seated(paco,10). seated(lucia,5). seated(antia,7). seated(jose,9). seated(martina,4).
       seated(claudia,2). seated(pedro,8)."
27 prueba_27 = "Comensales | seated(alberto,1). seated(carlos,3). seated(isabel,11). seated(juan,12).
       seated(carolina,6). seated(paco,10). seated(lucia,5). seated(antia,7). seated(jose,9).
       seated(martina,4). seated(claudia,2). seated(pedro,8)."
28 prueba_28 = "Comensales | seated(alberto,1). seated(carlos,3). seated(isabel,11). seated(juan,12).
       seated(carolina,6). seated(paco,10). seated(lucia,5). seated(antia,7). seated(jose,9).
       seated(martina,4). seated(claudia,2). seated(pedro,8). seated(javier,13). seated(julian,14).
       seated(herminia,15)."
29 prueba_29 = "Comensales | seated(eduardo,16). seated(amelia,17). seated(alberto,1). seated(ana,18).
       seated(elisa,19). seated(raul,20). seated(carlos,3). seated(isabel,11). seated(juan,12).
       seated(carolina,6). seated(paco,10). seated(lucia,5). seated(antia,7). seated(jose,9).
       seated(martina,4). seated(claudia,2). seated(pedro,8). seated(javier,13). seated(julian,14).
       seated(herminia,15)."
30 prueba_30 = "Comensales | seated(eduardo,16). seated(amelia,17). seated(alberto,1). seated(ana,18).
       seated(elisa,19). seated(raul,20). seated(carlos,3). seated(isabel,11). seated(juan,12).
       seated(carolina,6). seated(paco,10). seated(jesus,25). seated(amable,26). seated(dolores,27).
       seated(guillermo,28). seated(ivan,29). seated(sonia,30). seated(lucia,5). seated(antia,7).
       seated(jose,9). seated(marcos,21). seated(marcelino,22). seated(martina,4). seated(gabriel,23).
       seated(sergio,24). seated(claudia,2). seated(pedro,8). seated(javier,13). seated(julian,14).
       seated(herminia,15)."
```

B.3 Pruebas para *benchmark* de ganancia de sistema neurosimbólico

En tercer lugar, se presentan las 13 pruebas realizadas junto a sus puzzles para la prueba de ganancia del sistema neurosimbólico (sección 8.1.3).

```
1 prueba_1 = "Einstein | There are two houses, and two people: a German and a English. One house is
       painted Red and the other one is colored Purple. The German lives in the Red house."
```

```
2  prueba_2 = "Einstein | There are three houses. In them live a Spanish, an English and a Chinese. The
       houses are painted Red, Blue and Purple. There are three pets: a Dog, a Cat and a Duck. There
       are 3 tobacco brands: Ducados, Camel and Blue Master. There are also three drinks: Coffee, Water
       and Tea. The Spanish man house is painted Red, while the English man house is tainted Blue. The
       house of the Chinese man is Purple. The Spanish man keeps a Dog. The Chinese man loves to drink
       Tea. The Spanish one, on the other hand, smokes Ducados."
3  prueba_3 = "Einstein |  There are three houses. In them live a Spanish, an English and a Chinese.
       There are three pets: a Dog, a Cat and a Duck. The houses are painted Red, Blue and Purple.
       There are 3 tobacco brands: Ducados, Camel and Blue Master. There are also three drinks: Coffee,
       Water and Tea. The Spanish man house is painted Red, while the English house is tainted Blue.
       The Chinese man loves to drink Tea. The owner of the Purple house has a Duck pet. "
4  prueba_4 = "Einstein | There are three boats: one is Red, other is Purple and the other one is Green.
       There are a Spanish, an English and a Chinese. There are three drinks: Tea, Milk and Soda. The
       Spanish man drinks Soda. There's a Dog, a Cat and a Horse. The Spanish keeps the Cat, while the
       English man has a Dog. The Spanish man has a Red boat. The Chinese man, on the other hand, lives
       in a Purple boat."
5  prueba_5 = "Einstein | There are five houses: one is Yellow, other is Green, other is White, other is
       Red and the other is Blue. There's an British, a Danish, a German, a Norwegian and a Swedish.
       There are five pets: a Dog, a Fish, a Bird, a Horse and a Cat. The drinks are Beer, Coffee, Tea,
       Milk and Water. The tobacco brands are Blends, Blue Master, Dunhill, Pall Mall and Prince. The
       British lives in the Red house. The Swedish has a Dog. The Danish drinks Tea. The Green house is
       to the left of the White house. The owner of the Green house drinks Coffee. The person who
       smokes Pall Mall keeps a Bird. The owner of the Yellow house smokes Dunhill. The man living in
       the house number 3 drinks Milk. The Norwegian lives in the first house. The man who smokes
       Blends is neighbor of the one who keeps a Cat. The man who keeps a Horse lives next to the man
       who smokes Dunhill. The man who smokes Blue Master drinks Beer. The German smokes Prince. The
       Norwegian lives next to the Blue house. The man who smokes Blends has a neighbor who drinks
       Water."
6  prueba_6 = "Einstein | There are five houses: one is Gold, other is Olive, other is Ivory, other is
       Crimson and the other is Indigo. There's an Irish, a Welsh, a Scottish, a Belgian and a
       Moroccan. There are five pets: a Dragon, a Gryphon, a Wyvern, a Chimera and a Wyrm. The drinks
       present are Wine, Whiskey, Gin, Vodka and Vermut. The tobacco brands are Ducados, Camel,
       Winston, Pueblo and Lucky Strike. The Irish lives in the Crimson house. The Moroccan has a
       Dragon. The Welsh drinks Gin. The Olive house is to the left of the Ivory house. The owner of
       the Olive house drinks Whiskey. The person who smokes Pueblo keeps the Wyvern. The owner of the
       Gold house smokes Winston. The man living in the house number 3 drinks Vodka. The Belgian lives
       in the first house. The man who smokes Ducados is neighbor of the one who keeps a Wyrm. The man
       who keeps a Chimera lives next to the man who smokes Winston. The man who smokes Camel drinks
       Wine. The Scottish smokes Lucky Strike. The Belgian lives next to the Indigo house. The man who
       smokes Ducados has a neighbor who drinks Vermut."
7  prueba_7 = "Einstein | There are five houses whose colours are Gold, Olive, Ivory, Crimson and Indigo.
       In these houses live an Irish, a Welsh, a Scottish, a Belgian and a Moroccan. Five pets are
       kept: a Dragon, a Gryphon, a Wyvern, a Chimera and a Wyrm. Five drinks are drunk: Wine, Whiskey,
       Gin, Vodka and Vermut. The following brands are smoked: Ducados, Camel, Winston, Pueblo and
       Lucky Strike. The Crimson house is the home of the Irish. The Dragon is kept by the Moroccan.
       The Welsh's drink is Gin. The Ivory house is to the right of the Olive house. Whiskey is drunk
       in the Olive house. The Wyvern is kept in the house of the Pueblo smoker. The Winston smoker
       lives in the Gold house. Vodka is drunk in the third house. The Belgian lives in the house
       number 1. The Ducados smoker lives next to the one who keeps a Wyrm. The Chimera owner is
       neighbor of the Winston smoker. The Wine drinker is also a Camel smoker. The Scottish is a Lucky
       Strike smoker. The Indigo house is next to the house of the Belgian. The Ducados smoker has a
       neighbor who is a Vermut drinker."
8  prueba_8 = "Einstein | There are 20 houses. They are painted of the colors: Black, Blue, Purple,
       Magenta, Navy, Yellow, Ocre, White, Maroon, Ivory, Lime, Cyan, Rose, Pink, Grey, Green, Brown,
       Violet, Olive and Mustard. The people are a Dominican man, a Nicaraguan, a British, a Irish, a
       Norwegian, a Spanish, a Portuguese, a Chilean, a Peruvian, a Ecuatorian, an Italian, a German, a
       Belgian, a Swedish, a Swiss, a Greek, a Russian, an Ukranian, a Palestinian and a French."
9  prueba_9 = "Comensales | In this table there are 4 people sitting: Jose, Martina, Claudia and Pedro.
       The languages spoken in the table are German, Spanish and Arabic. Pedro and Jose speak Arabic.
       Claudia speaks Spanish and Martina speaks German."
10 prueba_10 = "Comensales | There are 3 people about to sit in a table: Juan, who speaks Spanish, Maria
       and Sofia. Both Maria and Sofia want to sit next to someone who speaks Spanish."
11 prueba_11 = "Comensales | In this table there are 4 people sitting: Jose, Martina, Claudia and Pedro.
       The languages spoken in the table are German, Spanish and Arabic. Pedro and Jose speak Arabic.
       Claudia speaks Spanish and Martina speaks German. Claudia refuses to sit next to Pedro. Pedro,
       on the other hand, wants to sit next to someone who speaks Arabic. Martina wants Jose to be on
       the opposite side of the table."
12 prueba_12 = "Comensales | In this table there are 10 people sitting: Alberto, Carlos, Carolina, Paco,
       Lucia, Isabel, Jose, Martina, Claudia and Pedro. The languages spoken in the table are German,
       Spanish, Galician, Catalan, Basque, French, English and Arabic. Pedro and Jose speak Arabic.
       Claudia speaks Spanish and Martina speaks German. Claudia refuses to sit next to Pedro. Pedro,
       on the other hand, wants to sit next to someone who speaks Arabic. Martina wants Jose to be on
```

```
    the opposite side of the table. Alberto only agrees to sit if Carolina is on the opposite side
    of the table. Isabel speaks French and wants to sit next to Pedro. Paco wants Lucía sitting on
    the opposite end of the table. Claudia wants to sit next to Carlos. Carolina speaks Basque and
    needs to sit next to someone who speaks French."
prueba_13 = "Comensales | In this table there are 25 people: A, B, C, D, E, F, G, H, I, J, K, L, M, N,
    O, P, Q, R, S, T, U, V, W, X and Y. The languages that are spoken are Spanish, Italian, German,
    Esperanto, Galician, English, Brazilian, Portuguese and French. A, D and C speak Brazilian. B, T
    and S are Spanish speakers."
```

Lista de acrónimos

AJAX Asynchronous JavaScript and Xml. 25, 27

API Application Programming Interface. 16, 25, 27, 30, 38, 50, 53

ASP Answer Set Programming. ii, iii, vi, 1, 2, 4, 5, 9–11, 15, 16, 18, 29–39, 44, 46–49, 53, 56–63

CNL Lenguaje Natural Controlado. 11

CSS Hojas de Estilo en Cascada. 18, 19

DL Deep Learning. 5

EE.UU Estados Unidos. 9

GB Gigabytes. 30

HTML Lenguaje de Marcas de Hipertexto. 18, 19

IA Inteligencia Artificial. 5

LLM Large Language Model. 1–3, 5, 7, 9–11, 15, 16, 18, 29, 30, 40, 50, 53, 72

LN Lenguaje Natural. ii, iii, 1, 2, 10, 11, 29, 30, 32, 40, 46, 47, 53, 56–63

NLP Procesamiento de Lenguaje Natural. 10

RAM Random-Access Memory. 30

REGEX Regular Expression. 34

REST REpresentational State Transfer. 27

RNA Red de Neuronas Artificiales. 5

SFT Supervised Fine-Tuning. 7

TFG Trabajo de Fin de Grado. 1

VRAM Video Random-Access Memory. 30

XML Lenguaje de Marcas Extendible. 18

Glosario

7B Gran Modelo de Lenguaje que ha sido entrenado con aproximadamente 7 mil millones de parámetros, correspondientes a billones americanos (de ahí la letra). Este límite (así como algunos superiores, como 13B, 33B, 70B), fue determinado en el artículo de Llama *"LLaMA: Open and Efficient Foundation Language Models"*[26] como un método de sesgar diferentes modelos en base a su presupuesto en términos de capacidad inferencial.. 30

backend Dicho de la parte del desarrollo de un sistema informático centrada en la parte oculta al usuario, sean redes, algoritmia, *scripts* o análogos.. 27, 28

benchmark Punto estándar de comparación sobre el que diferentes elementos pueden ser evaluados. En informática, comúnmente, procedimiento algorítmico sobre condiciones fijas que permiten puntuar productos, sistemas o algoritmos determinados.. 46

Byte En computación, unidad de capacidad formada por 8 unidades mínimas binarias (*bits*). 30

dataset Colección de datos ordenados con una relación temática, metodológica o unidos por alguna adyacencia de cualquier tipo o interés común.. 11

endpoint Punto de contacto expuesto por un sistema para recibir comunicaciones por parte de un elemento externo en términos lógicos. plural. 28

framework Abstracción de un programa software que proporciona una interfaz para especializar y automatizar su uso para determinados cometidos. plural. 19, 27

frontend Dicho de la parte del desarrollo de un sistema informático centrada en la interfaz gráfica o experiencia de usuario.. 3, 18, 25, 28

hardware Elementos físicos que forman parte de un computador y conforman al aparato en lo mecánico.. 30, 49

macro Conjunto de instrucciones o funcionalidades simplificadas en una sóla ejecución para facilitar determinada labor. plural. 5

parser Analizador sintáctico que recorre secuencias de lenguaje natural con el objetivo de procesarlo o realizar sobre él cualquier tarea con incidencia en su sintaxis.. 10

software Se dice de aquellos elementos de un computador no tangibles, basados en reglas lógicas, programas o algoritmos.. 5, 12, 17, 29

solver Herramienta software de programación lógica que permite generar conjuntos de soluciones dado un sistema lógico de variables y reglas que las rigen.. 2

token Forma numérica de representación de texto sobre la cual se entrenan los LLM. Es la forma mediante la cual un modelo ajusta los pesos de la probabilidad entre diferentes palabras o conjuntos de palabras. plural. 5, 6, 30

Bibliografía

[1] V. D. Kirova, C. S. Ku, J. R. Laracy, and T. J. Marlowe, "Software engineering education must adapt and evolve for an llm environment," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 666–672. [En línea]. Disponible en: https://doi.org/10.1145/3626252.3630927

[2] G. Brewka, T. Eiter, and M. Truszczynski, "Answer set programming at a glance," *Commun. ACM*, vol. 54, pp. 92–103, 12 2011.

[3] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Answer set solving in practice," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, pp. 1–238, 12 2012.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [En línea]. Disponible en: https://arxiv.org/abs/1706.03762

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[6] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. Raffel, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," 2022. [En línea]. Disponible en: https://arxiv.org/abs/2205.05638

[7] M. McMullen, "Choosing the Right Technique: Prompt Engineering vs Fine-tuning — datasciencecentral.com," https://www.datasciencecentral.com/choosing-the-right-technique-prompt-engineering-vs-fine-tuning/, [Accessed 22-04-2024].

[8] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022.

[9] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, N. Joseph, S. Kadavath, J. Kernion, T. Conerly, S. El-Showk, N. Elhage, Z. Hatfield-Dodds, D. Hernandez, T. Hume, S. Johnston, S. Kravec, L. Lovitt, N. Nanda, C. Olsson, D. Amodei, T. Brown, J. Clark, S. McCandlish, C. Olah, B. Mann, and J. Kaplan, "Training a helpful and harmless assistant with reinforcement learning from human feedback," 2022.

[10] "reasoning-Publications — azreasoners.github.io," https://azreasoners.github.io/ARG-webpage/publication.html, [Accessed 05-04-2024].

[11] Z. Yang, A. Ishay, and J. Lee, "Coupling large language models with logic programming for robust and general reasoning from text," 2023.

[12] A. Ishay, Z. Yang, and J. Lee, "Leveraging large language models to generate answer set programs," 2023.

[13] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 1–14, 12 2020.

[14] M. Chen, Y. Ma, K. Song, Y. Cao, Y. Zhang, and D. Li, "Learning to teach large language models logical reasoning," 2023.

[15] A. Creswell, M. Shanahan, and I. Higgins, "Selection-inference: Exploiting large language models for interpretable logical reasoning," 2022.

[16] M. Borroto, I. Kareem, and F. Ricca, "Towards automatic composition of asp programs from natural language specifications," 2024.

[17] T. Kuhn, "A survey and classification of controlled natural languages," *Computational Linguistics*, vol. 40, no. 1, pp. 121–170, Mar. 2014. [En línea]. Disponible en: https://aclanthology.org/J14-1005

[18] S. CARUSO, C. DODARO, M. MARATEA, M. MOCHI, and F. RICCIO, "Cnl2asp: Converting controlled natural language sentences into asp," *Theory and Practice of Logic Programming*, vol. 24, no. 2, p. 196–226, 2024.

[19] A. Rajasekharan, Y. Zeng, P. Padalkar, and G. Gupta, "Reliable natural language understanding with large language models and answer set programming," *Electronic*

Proceedings in Theoretical Computer Science, vol. 385, p. 274–287, Sep. 2023. [En línea]. Disponible en: http://dx.doi.org/10.4204/EPTCS.385.27

[20] Y. Zeng, A. Rajasekharan, P. Padalkar, K. Basu, J. Arias, and G. Gupta, "Automated interactive domain-specific conversational agents that understand human dialogs," 2023.

[21] Y. Ding, X. Zhang, C. Paxton, and S. Zhang, "Leveraging commonsense knowledge from large language models for task and motion planning," in *RSS 2023 Workshop on Learning for Task and Motion Planning*, 2023. [En línea]. Disponible en: https://openreview.net/forum?id=LMiTmpZgSZ

[22] D. Yu, B. Yang, D. Liu, H. Wang, and S. Pan, "A survey on neural-symbolic learning systems," 2023.

[23] P. Tarau, "Full automation of goal-driven llm dialog threads with and-or recursors and refiner oracles," 2023.

[24] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "Llm-assisted generation of hardware assertions," 2023.

[25] "GitHub - oobabooga/text-generation-webui: A Gradio web UI for Large Language Models. Supports transformers, GPTQ, AWQ, EXL2, llama.cpp (GGUF), Llama models. — github.com," https://github.com/oobabooga/text-generation-webui, [Accessed 05-04-2024].

[26] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.