ProgrEval: Generación asistida de consignas de evaluación para la enseñanza de programación

Tomás Caballero[1](0009-0009-3170-3829), Agustín Fernández-Ortúzar[1](0009-0007-0685-6239), Gonzalo Pablo Fernández[1](0009-0001-6146-8462) y Christian Cossio-Mercado[1,2](0000-0002-2535-3078)

[1] Universidad de Buenos Aires. Facultad de Ciencias Exactas y Naturales. Departamento de Computación. Buenos Aires, Argentina.
{tcaballero,afernandezortuzar,gpfernandez,ccossio}@dc.uba.ar
[2] CONICET-Universidad de Buenos Aires. Instituto de Ciencias de la Computación (ICC). Buenos Aires, Argentina.

Resumen El diseño de evaluaciones constituye una tarea central en el ámbito educativo. En particular, la evaluación en programación presenta desafíos adicionales, ya que no solo requiere valorar la comprensión conceptual y la producción de código, sino también la formulación de problemas, la depuración y la explicación o seguimiento de programas. La elaboración de estas consignas suele realizarse manualmente, lo que demanda tiempo, experiencia disciplinar y criterios pedagógicos sólidos. La generación automática de ítems (Automatic Item Generation, AIG) surgió como una línea de investigación orientada a asistir la creación de evaluaciones. Tradicionalmente, se ha centrado en plantillas con parámetros predefinidos, especialmente para cuestionarios de opción múltiple. Aunque estos enfoques ofrecen control, presentan limitaciones para adaptarse a distintos objetivos de evaluación y a la diversidad de desempeños propios de la enseñanza de la programación. En este trabajo se presenta ProgrEval, un software que asiste en la generación automática de consignas de programación mediante el uso de grandes modelos de lenguaje (LLM). El sistema releva los requerimientos de evaluación del docente y utiliza una ontología que representa relaciones entre conceptos, desempeños, niveles de complejidad y formatos de actividad. A partir de esta información, genera *prompts* estructurados que formalizan y expanden el requerimiento inicial para su uso en herramientas de inteligencia artificial generativa. De esta manera, ProgrEval facilita la producción de consignas pertinentes y alineadas con los objetivos pedagógicos definidos por el docente.

Palabras claves: Evaluación en programación; Generación automática de ítems; Grandes modelos de lenguaje; Ontologías educativas; Educación en informática

ProgrEval: Assisted Generation of Assessment Tasks for Programming Education

Abstract. Assessment design is a central task in education. In programming education, assessment presents additional challenges, since it requires evaluating not only conceptual understanding and code production, but also problem formulation, debugging, and program explanation or tracing. The creation of these tasks is usually carried out manually by instructors, demanding significant time, strong disciplinary expertise, and well-defined pedagogical criteria. Automatic Item Generation (AIG) emerged as a research area focused on supporting assessment creation. Traditionally, AIG approaches have relied on templates with predefined parameters, especially for multiple-choice questionnaires. Although these methods provide a high degree of control, they present limitations when adapting to different assessment objectives and to the variety of performances involved in programming education. In this work, we present ProgrEval, a software tool that assists in the automatic generation of

programming assessment tasks using Large Language Models (LLMs). The system gathers the instructor's assessment requirements and uses an ontology that represents relationships among concepts, performances, complexity levels, and activity formats. Based on this information, it generates structured prompts that formalize and expand the initial requirement for use with artificial intelligence tools. In this way, ProgrEval facilitates the production of relevant assessment tasks aligned with the pedagogical objectives defined by the instructor.

Keywords: Programming Assessment; Automatic Item Generation; Large Language Models; Educational Ontologies; Computing Education

1 Introducción

El diseño de evaluaciones constituye una tarea central en el ámbito educativo, ya que permite obtener evidencias sobre el nivel de comprensión y desempeño de los estudiantes. En particular, la evaluación de conceptos de programación presenta desafíos adicionales si se considera, además de los conceptos y la habilidad de producir código, otros desempeños que pueden poner en juego los estudiantes, como la especificación de problemas, la depuración, la explicación o el seguimiento de programas. Por otro lado, la elaboración de este tipo de consignas es, en la mayoría de los casos, realizada manualmente por los docentes, requiriendo una gran inversión de tiempo, experiencia disciplinar y criterios pedagógicos.

En este contexto, la generación automática de ítems (en inglés, *Automatic Item Generation*, AIG) surgió como una línea de investigación orientada al diseño de herramientas que asistan en la creación de evaluaciones. Tradicionalmente, los trabajos de AIG se han enfocado en el uso de plantillas para generar consignas variando datos parametrizados y su implementación en cuestionarios de opción múltiple. Si bien estos enfoques permiten manejar la generación con un alto grado de control, presentan limitaciones en términos de capacidad para adaptarse a distintos objetivos de evaluación, especialmente ante la necesidad de evaluar una variedad de conceptos y desempeños utilizando distintos formatos de actividad.

El desarrollo de grandes modelos de lenguaje (Large Language Models, LLMs) y la disponibilidad de agentes conversacionales habilitó la exploración de nuevos enfoques para la generación automática de consignas, principalmente basadas en ingeniería de *prompts*. Estos modelos abren la posibilidad de introducir contextos permitiendo la generación de consignas más diversas y adaptadas a distintos escenarios educativos. A pesar de eso, el comportamiento de estos modelos depende, en gran medida, de la estructura y organización del contenido en los prompts, y los enfoques actuales proponen estrategias *ad hoc*, sin una estructura formal que garantice la alineación entre los objetivos de la evaluación y las consignas generadas.

En este trabajo proponemos un enfoque para la generación automática de consignas de programación que integra una ontología con el diseño de *prompts* estructurados para modelos de lenguaje. La ontología permite representar relaciones entre los elementos didácticos principales de una evaluación, como conceptos, desempeños, niveles de complejidad y formatos de actividad, mientras que el *prompt* tiene por objetivo actuar como interfaz para sistematizar y proveer información estructurada a un modelo de lenguaje. Este enfoque busca desacoplar la definición de los requerimientos de la evaluación de su materialización en una consigna, permitiendo generar *prompts* automáticamente que guíen la producción de consignas alineadas con los requerimientos.

Los aportes principales de este trabajo son el diseño de una ontología para la representación de evaluaciones de programación, la definición de una plantilla estructurada de *prompts* y una herramienta online que automatiza la generación de prompts, permitiendo que un docente pueda enfocar los esfuerzos de diseño de evaluaciones en la definición de los criterios pedagógicos.

2 Trabajos relacionados

El campo de la **generación automática de ítems** (*Automatic Item Generation*, AIG) comenzó a desarrollarse en la década de los 70 con un trabajo de John B. Bormuth, en el que propone un enfoque para generar preguntas a partir de pasajes en prosa mediante métodos sintácticos (Bormuth y Menzel, 1970; Gierl y Lai, 2016).

Históricamente, el diseño y confección de evaluaciones ha sido un proceso manual que involucra, principalmente, dos etapas: una de **decisiones didácticas** (¿qué queremos evaluar?) y otra **creativa**, que implica la elaboración de las consignas, la definición del dominio del problema y, en algunos casos, el desarrollo de una narrativa. Sin embargo, llevar adelante este proceso de manera efectiva, requiere una inversión considerable de tiempo, así como experiencia disciplinar y conocimientos psicométricos por parte de los docentes (Burke, 2025).

En este contexto, la generación automática cobró relevancia en los últimos años. Un análisis de 71 trabajos publicados desde el 2010 hasta al 2024 evidencia un crecimiento en el uso de estas técnicas en el ámbito educativo (Song et al., 2025), siendo las consignas de opción múltiple (*Multiple Choice Question*, MCQ) el formato de actividad más utilizado.

Dentro de las técnicas tradicionales de AIG, el uso de **plantillas** ha sido uno de los enfoques más extendidos. Una plantilla define la estructura de una consigna, identificando partes con contenido estático y otras partes con datos parametrizados. Para obtener una consigna resoluble, se completan automáticamente los datos parametrizados, tomando valores de un conjunto o rango posible. Por ejemplo, una consigna de matemáticas puede generarse a partir de una consigna donde la ecuación involucrada es una expresión parametrizada y los argumentos posibles tienen rangos previamente definidos:

> *Supongamos que, al despegar, la distancia recorrida por un avión de pasajeros en la pista se puede determinar mediante la siguiente ecuación en metros:*
> $D = a * S^2 + b * S$
> *Donde S es el número de segundos que ha transcurrido desde el despegue.*
> *¿Qué distancia habrá recorrido el avión después de c segundos?*

En este ejemplo, los parámetros del template son **a**, **b** y **c**. Para completar los valores y generar una consigna automáticamente, podríamos considerar las siguientes restricciones:

- **a** es un valor entero en el rango: 5 – 9
- **b** es un valor entero en el rango: 2 – 9
- **c** es un valor entero en el rango: 10 – 20

Si bien las técnicas basadas en plantillas permiten generar consignas de manera sistemática y controlada, podemos identificar dos limitaciones principales. Por un lado, este enfoque **restringe la variabilidad** de las consignas a las combinaciones posibles de los parámetros definidos en la estructura, lo que puede derivar en consignas repetidas o muy similares. Por otro lado, la confección de nuevas plantillas **requiere un esfuerzo manual** por parte de quien las quiera utilizar, tanto para definir las estructura principal (contenido estático y parámetros) como las restricciones sobre los parámetros.

En el contexto de evaluaciones de conocimientos de programación, en el que se buscó elaborar consignas en diferentes formatos para que los estudiantes pongan en juego diferentes desempeños, estas limitaciones se vuelven más evidentes. En particular, este tipo de técnicas dificulta la generación de consignas que evalúen distintos desempeños, como la implementación, la depuración o la explicación, ya que cada uno de estos requiere estructuras de consigna significativamente diferentes. Por lo tanto, estas limitaciones motivan la exploración de enfoques más flexibles, como aquellos basados en modelos de lenguaje.

Con el avance reciente de los grandes modelos de lenguaje (*Large Language Models*, LLMs), surgieron enfoques basados en la ingeniería de *prompts* (*prompt engineering*) para la genera-

ción de consignas en diversas disciplinas como física (Omopekunola y Kardanova, 2024), programación (Doughty et al., 2024) y disciplinas STEM en general (Chan et al., 2025). El objetivo no es emplear estas herramientas para resolver directamente la tarea de generación sino semi-automatizar la generación de evaluaciones y centrar el proceso de diseño en los objetivos didácticos. Asimismo, si estas herramientas producen consignas de calidad, es posible reducir significativamente el tiempo y el esfuerzo para su elaboración.

Sin embargo, la exploración actual del uso de la ingeniería de *prompts* para AIG en el ámbito educativo es insuficiente. Más allá de escribir y modificar prompts, ciertos aspectos pueden mejorarse, por ejemplo, incorporando ontologías en el flujo de trabajo para que los enfoques sean teóricamente sólidos. (Song et al., 2025)

3 *ProgrEval*: una herramienta comprensiva para la generación de consignas

Al momento de diseñar una evaluación entran en tensión dos aspectos: la definición de lo que se quiere evaluar y la producción de las consignas, siendo el primero lo más relevante porque establece las evidencias de aprendizaje que interesa recolectar de los estudiantes y los criterios que se van a tener en cuenta para diseñar la consigna. Con el objetivo de facilitar el proceso de confección de consignas poniendo el foco en los requisitos didácticos de la evaluación, se diseñó *ProgrEval*, una herramienta que toma requisitos acerca de lo que se quiere evaluar y, a partir de una base de conocimiento, devuelve consignas de ejemplo, alineadas con los requisitos elegidos, y un texto estructurado que puede ser utilizado como entrada de un modelo de lenguaje para generar consignas automáticamente.

3.1 Descripción de la herramienta

La herramienta *ProgrEval* está compuesta de dos partes integradas en una interfaz web:

- Una **base de conocimiento**, implementada sobre una ontología, donde se organizan y relacionan los conceptos disciplinares, los desempeños de programación que los estudiantes pueden poner en juego, consignas de ejemplo, entre otros elementos.
- Un **generador de descripciones** que combina los requisitos ingresados de manera organizada y estructurada para ser utilizados en un modelo de lenguaje, como *ChatGPT*[1] o *Gemini*[2].

Estructura de la ontología En este trabajo proponemos el diseño de una ontología para representar el dominio de las consignas de evaluación de programación. En el contexto de las Ciencias de la Computación, una **ontología** es una representación formal y explícita de un dominio de conocimiento que describe sus conceptos, relaciones y restricciones de manera estructurada. A partir de esta representación, es posible realizar consultas que aprovechen las relaciones definidas e inferir nuevas relaciones mediante mecanismos de razonamiento automático. Su objetivo es albergar consignas modelo de todos los tipos posibles en el campo de la evaluación de los aprendizajes de programación. Para eso, definimos una organización en tres ejes: **Concepto**, **Desempeño** y **Formato de actividad**. Los primeros dos definen evidencias de *cuáles* son los aprendizajes que se buscan, y el tercero define *cómo* se obtienen.

Como se ve en la Figura 1, cada **consigna modelo** es ejemplo de un **tipo de consigna**, que asocia una tripla de **concepto**, **desempeño** y **formato de consigna**. A cada una le corresponden además los **públicos objetivo** a los que apuntan, el **nivel de complejidad** con el que evalúan el

[1] OpenAI ChatGPT - Sitio web: https://chatgpt.com
[2] Google Gemini - Sitio web: https://gemini.google.com/app

concepto, y las **habilidades** requeridas como conocimiento previo (entendiendo como habilidad a la capacidad de emplear un desempeño sobre un concepto particulares). El rol que cumplen cada uno de estos elementos se detalla más adelante.

Para que *ProgrEval* pueda usarse en cualquier curso introductorio de programación, debe ser independiente de cualquier lenguaje de programación. Por eso, todos los elementos de lenguaje, como los fragmentos de código pertenecientes a las consignas, se expresan en pseudocódigo. Sin embargo, algunas consignas pueden contener problemas que están definidos dentro de un escenario particular (por ejemplo, un tablero o una consola de comandos) o como un tipo de problema particular (por ejemplo, como un problema de transformación de estados o de transformación de información) . Actualmente, gran parte de las consignas almacenadas en la ontología provienen de guías de ejercicios de la materia *Introducción a la Programación* de la Universidad Nacional de Quilmes elaboradas para ser resueltas utilizando el lenguaje *Gobstones*[3], en el que se trabaja sobre un tablero con bolitas de colores y un cabezal que se puede mover sobre las celdas y operar con bolitas. Para lograr la independencia del lenguaje, los ejercicios agregados a la ontología fueron analizados manualmente y adaptados para no incluir palabras reservadas de *Gobstones*. Por ejemplo, en este caso únicamente fue necesario adaptar la solución:

Ejercicio de Gobstones y su solución:	**Adaptación del ejercicio y su solución:**
Consigna: Escribir el procedimiento Poner6Negras() que ponga 6 bolitas de color Negro en la celda actual. Respuesta posible: procedure Poner6Negras() { repeat (6) { Poner(Negro) } }	Consigna: Escribir el procedimiento Poner6Negras() que ponga 6 bolitas de color Negro en la celda actual. Respuesta posible: FUNCION Poner6Negras() { REPETIR 6 VECES { Poner(Negro) } }

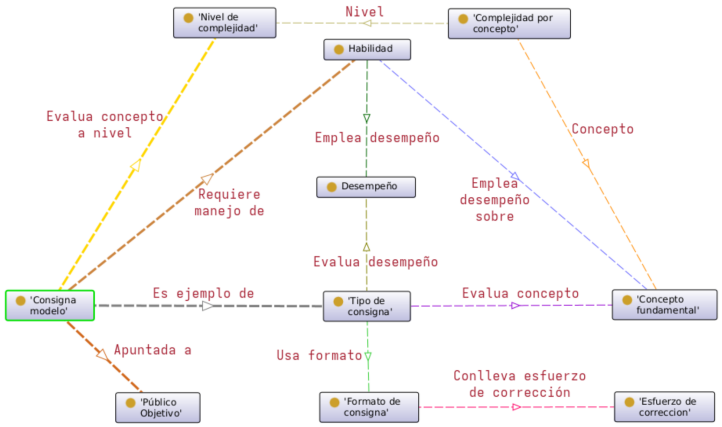

Figura 1. Representación de la ontología.

Generación automática de ítems utilizando *prompt engineering* El diseño de *prompts* ha sido ampliamente explorado durante los últimos años, particularmente a partir del desarrollo y disponibilidad de grandes modelos de lenguaje como *ChatGPT*, *Gemini* y *DeepSeek*[4], por mencionar algunos. Diversos trabajos dieron cuenta de que la forma en que se estructura un *prompt* tiene un impacto significativo en la calidad, coherencia y adecuación de las respuestas generadas (Khoirul Anam, 2025).

En este contexto, han surgido diferentes estrategias de diseño de *prompts*, bajo el nombre de *prompt engineering*, que buscan guiar el comportamiento de los modelos de lenguaje. Por

[3] Sitio web de Gobstones: gobstones.github.io
[4] Deepseek AI chat: chat.deepseek.com

ejemplo, los enfoques basados en la definición explícita de instrucciones (*instruction prompting*) y el uso de ejemplos (*few-shot prompting*), donde se proveen ejemplos de entrada y salida esperada para orientar la generación (Chan et al., 2025; Doughty et al., 2024; Sarsa et al., 2022).

Asimismo, distintos trabajos (Chan et al., 2025; Doughty et al., 2024) y guías prácticas[5] proponen organizar la estructura de los *prompts* en secciones que cumplan un propósito específico dentro del proceso de generación. En general, podemos identificar partes como: la definición del **rol** o **identidad** que debe asumir el modelo, un conjunto de **instrucciones precisas** sobre la tarea a realizar, información del **contexto** y **ejemplos** que ilustran el tipo de salida esperada. Esta organización pretende favorecer una mayor alineación entre los requerimientos de un docente y las consignas generadas por un modelo.

Tomando como referencia estos enfoques, y a partir de los requerimientos de la consigna a generar, se diseñó una plantilla de *prompt* estructurado para generar nuevas consignas mediante un modelo de lenguaje. Dada la estructura del *prompt* y los requisitos de la consigna, la herramienta genera automáticamente un *prompt* que además es enriquecido con la ontología, incorporando los ejemplos y definiciones de la terminología, con el objetivo de garantizar un grado alto de coherencia entre las consignas generadas y los criterios definidos previamente.

La plantilla del *prompt* está organizada en las siguiente secciones:

- **Identidad**: Se especifica el rol de experto en evaluaciones de programación y el nivel educativo en el que se va a trabajar.
- **Terminología**: Definiciones tomadas de la ontología de los elementos involucrados para generar la consigna. Incluye información sobre *Conceptos*, *Desempeños*, *Conocimientos previos*, *Niveles de complejidad* y *Formato de Actividad*.
- **Tarea**: Requisitos de la evaluación a generar, explicitando el concepto y desempeño que se quiere evaluar, y una lista de ejemplos alineados con los objetivos indicados. A su vez, esta sección especifica los **requisitos de la consigna** agrupando los requerimientos pedagógicos indicados y **ejemplos de consignas** alineadas con estas definiciones.
- **Formato de salida**: Precisiones acerca de la salida esperada, indicando el formato del texto y la inclusión de soluciones.

De esta manera, el *prompt* resultante constituye una representación estructurada de los requerimientos de la consigna, priorizando la definición de los objetivos didácticos e información del contexto en el que será utilizada.

Uso de la herramienta Una prueba de concepto de la herramienta está disponible online, a través de progreval.dc.uba.ar.

Al utilizarla, el docente debe completar todos sus campos obligatorios, pensados para ayudar a identificar precisamente **qué se quiere evaluar** y **en qué contexto**. Una vez que se ingresan los requisitos, utilizando el botón “Mostrar ejemplos” la herramienta consulta y recupera de la ontología al menos una consigna modelo diseñada previamente siguiendo los mismos requisitos ingresados, garantizando un alto grado de validez. A partir de estos ejemplos, se espera que un docente pueda adaptarlos manualmente, o bien, utilizar la descripción (*prompt*) que devuelve la herramienta para generar consignas estructuralmente equivalentes con un modelo de lenguaje.

Los primeros requisitos que se solicitan son el **Público objetivo** (que limita el dominio de los problemas a generar) y las **intenciones didácticas** de la evaluación (el concepto, el nivel de complejidad y el desempeño que se pone en juego). Las opciones disponibles para el público objetivo son: **Nivel inicial**, **Nivel primario** (primer y segundo ciclo), **Nivel secundario** (primer y segundo ciclo) y **Nivel Universitario**. Al determinar este requisito, queremos evitar que el

[5] Ejemplos de guías: cloud.google.com/discover/what-is-prompt-engineering y developers.openai.com/api/docs/guides/prompt-engineering

Figura 2. Pantalla principal de ProgrEval y los primeros requisitos que se solicitan.

dominio de la consigna agregue complejidad y no permita cumplir el propósito de la evaluación. Por ejemplo, evitar que una consigna de primer ciclo de primaria hable de un sistema bancario.

Los siguientes requisitos a seleccionar son el **Concepto** principal y el **Nivel de complejidad** con el que se lo quiere evaluar. El nivel de complejidad se define en términos de las maneras en que se puede emplear un concepto para resolver una consigna y de cómo se combina con otros conceptos. Por ejemplo, una consigna puede requerir la repetición de una secuencia de instrucciones una cantidad fija de veces (nivel básico), mientras que otras consignas pueden requerir que una secuencia de instrucciones se repita mientras se cumplan ciertas condiciones (intermedio).

Para facilitar la verificación de validez de una consigna, se trabajó sobre consignas que evalúan un concepto específico y pone en juego un desempeño de programación particular. La lista de conceptos utilizada está compuesta por conceptos fundamentales de programación que se recopilaron de distintos trabajos (Grover, 2020; Tew y Guzdial, 2010), como repetición, alternativa condicional, variables, funciones, expresiones (lógicas y aritméticas) y tipos de datos. Esta lista no busca ser exhaustiva y tiene como propósito definir una base mínima de conceptos que pueden formar parte de un curso inicial de programación. En la práctica, los conceptos pueden ir variando en cantidad, profundidad y dificultad según la currícula de cada materia y el nivel educativo.

Para establecer concretamente de qué se habla cuando se menciona un concepto, proveemos una definición para cada uno. Por ejemplo, el concepto **"directiva de repetición"** utilizado es el siguiente:

Directiva de repetición: *Directivas provistas por los lenguajes de programación para definir la repetición de secuencias de cómputo. En la mayoría de los lenguajes imperativos podemos definir bucles usando directivas como while y for.*

Los niveles de complejidad asociados a este concepto están definidos de la siguiente manera:

Niveles de complejidad:

- *Básico*: Repetición simple, cantidad fija de iteraciones
- *Intermedio*: Repetición condicional o repetición simple anidada
- *Avanzado*: Repetición condicional anidada

Además del concepto que se quiere evaluar y el nivel de complejidad, una característica que se considera relevante y prioritaria al momento de definir los requisitos de una consigna es el **desempeño de programación** que queremos que los estudiantes pongan en juego al resolverla. En trabajos previos sobre generación automática de evaluaciones ((Doughty et al., 2024; Omopekunola y Kardanova, 2024)) consideran desempeños alineados de manera directa con la taxonomía de Bloom(Atherton, 2011), como recordar, analizar, aplicar y evaluar. Para nuestro trabajo, definimos un conjunto de desempeños alineados con objetivos de programación, como implementación, seguimiento, depuración y especificación, entre otros (Bower, 2008; Ruf et al., 2015; Simões y

Queirós, 2020). Al igual que con los conceptos, para cada desempeño proveemos una definición en el contexto de programación, por ejemplo, *"**Implementación**: Habilidad para diseñar y construir un programa que cumpla con un objetivo dado."*.

El siguiente requisito a considerar son los **conocimientos previos** que poseen los estudiantes a ser evaluados. Si no se asumen ciertos conocimientos como ya adquiridos, se podría estar diseñando consignas que priorizan otros conceptos o introducen conceptos no abordados, dificultando la interpretación de la evidencia que se busca recolectar con la evaluación. Por lo tanto, éstos serán determinantes en el modo en que el concepto principal se puede integrar con otros sin perder su protagonismo y sin que se desdibuje la noción de lo que se está evaluando. Para seleccionarlos, la herramienta presenta una tabla de doble entrada sobre la cual el usuario puede seleccionar qué desempeños considera que sus estudiantes ya dominan para cada concepto disponible (ver Figura 3). Por ejemplo, si al momento de generar las consignas ya se abordaron los conceptos de expresión matemática y función, se podrían seleccionar los casilleros **"Definición-Expresión matemática"**, **"Definición-Función"** e **"Implementación-Función"**.

El último paso que termina de definir los requerimientos de la consigna es la elección de un **formato de actividad**. Al momento de elegir, el docente debe ponderar el tipo de evidencia que quiere recolectar de los resultados contra las condiciones prácticas que lo limiten. Si se piensa en obtener los resultados que más información brindan acerca del nivel actual del estudiante, una consigna de **respuesta abierta** (definida como "*Consigna para la que hay que desarrollar una de sus posibles respuestas desde cero. Su corrección requiere criterios graduales o rúbricas, ya que las respuestas pueden diferir en forma y calidad.*") podría ser la más adecuada pero requiere más inversión de tiempo en su resolución y corrección contra otros formatos como un cuestionario de opción múltiple.

Para facilitar esta elección, se le asignó a cada formato un **esfuerzo de corrección**, dentro de un rango que va desde la corrección automática (bajo esfuerzo) hasta una revisión manual de las respuestas (alto esfuerzo). Para este requisito entendemos que hay una correlación directa con el esfuerzo de resolución, y una correlación inversa con la calidad de la evidencia obtenible. Actualmente, lo dividimos en cuatro niveles: Automático (e.g. **Opción múltiple**), Semi-Automático (e.g. **Compleción de espacios vacíos**), Moderado (e.g. **Verdadero o falso con justificación**) e Intenso (e.g. **Respuesta abierta**).

En la Figura 4 se muestra una salida posible que puede generar la herramienta si se definen los siguientes requisitos:

- Público objetivo: Nivel universitario.
- Concepto: Directiva de repetición.
- Nivel de complejidad: Intermedio.
- Desempeño: Implementación.
- Conocimientos previos: (Definición, Expresión matemática), (Definición, Función), (Implementación, Función).
- Formato de actividad: Respuesta abierta.

Por último, la herramienta permite generar un *prompt* que utiliza la estructura definida en la Sección 3.1 e incluye los requisitos de la consigna y los ejemplos obtenidos de la ontología, listo para ser utilizado en un modelo de lenguaje y generar consignas automáticamente, alineadas con los objetivos didácticos.

Para añadir detalles específicos al contexto en que se desea usar la consigna generada, el *prompt* incorporará el contenido de los campos de **Contexto específico** (ver Figura 5). Estos campos buscan atender las áreas que la ontología no cubre, como el lenguaje de programación, características del mismo que se desea omitir, reglas de buenas prácticas y cualquier información extra que el docente vea pertinente incluir.

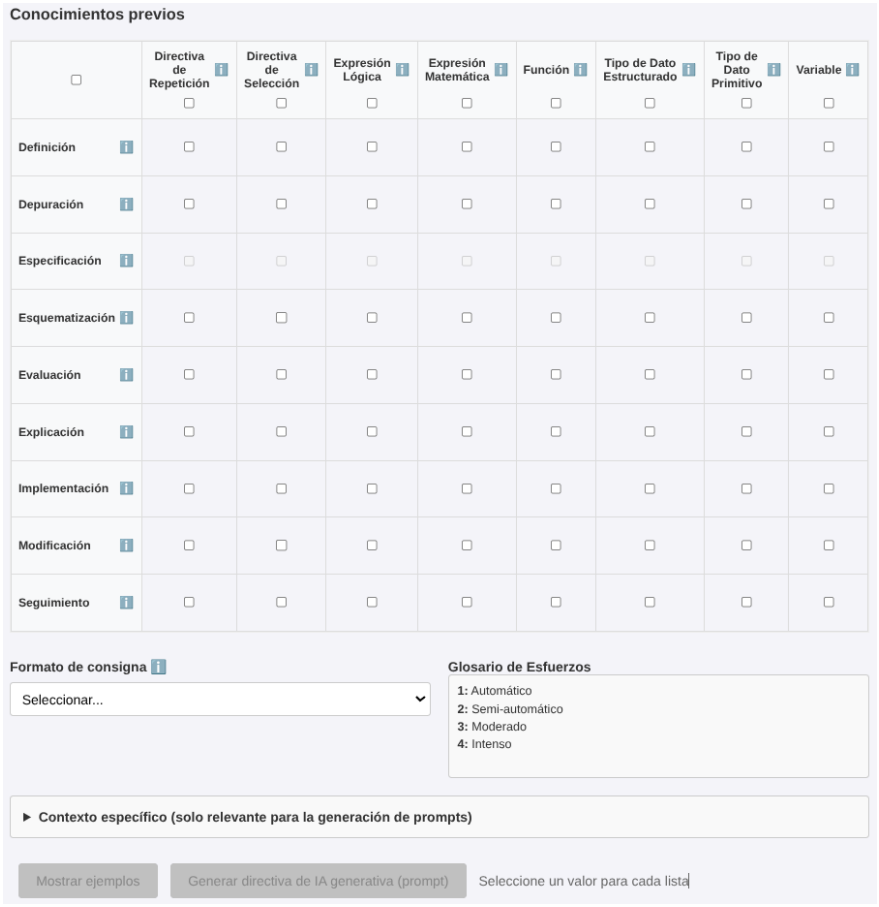

Figura 3. Pantalla principal de ProgrEval.

4 Validación de la herramienta

Para validar este desarrollo, interesa medir la calidad y la validez de las consignas generadas utilizando tres tipos de *prompts* diferentes (el *prompt* diseñado y dos variantes más) en los agentes conversacionales *Gemini* y *ChatGPT*, buscando responder las siguientes preguntas:

- ¿En qué medida las consignas generadas están alineadas con los requerimientos didácticos especificados?
- ¿En qué medida las consignas generadas presentan una alta calidad?

4.1 Diseño del experimento

Para limitar la cantidad de requerimientos didácticos a probar, realizamos la siguiente selección: 2 conceptos de programación (repetición y expresiones lógicas), una combinación de 5 pares desempeño/formato-de-actividad por concepto, el nivel de complejidad básico para repetición y nivel de complejidad intermedio para expresiones lógicas. A partir de esta selección, priorizamos tener una variedad de formatos de actividad por sobre la cantidad de conceptos a evaluar. El resultado es un total de 10 configuraciones posibles que se ingresaron en *ProgrEval* para generar automáticamente los *prompts* asociados.

Figura 4. Consigna de ejemplo obtenida con ProgrEval, teniendo en cuenta los requisitos ingresados.

Además del *prompt* diseñado, consideramos dos modelos de *prompt* adicionales: el *prompt* diseñado pero sin la información de la ontología y un modelo de *prompt* sin estructura. A partir de esto y considerando las 10 configuraciones de requerimientos elegidas, confeccionamos un conjunto de 30 *prompts* para generar las consignas. Luego, utilizamos los prompts en los modelos *GPT-5.5 Instant* y *Gemini Fast 3*, obteniendo un total de 60 estímulos para analizar. Estos resultados fueron generados utilizando la versión online de cada agente y configurados con conversaciones temporales para evitar que los modelos incorporen información adicional de contexto que pueda influir en el resultado final.

4.2 Participantes

Para realizar el experimento participaron 5 docentes con conocimientos y experiencia en materias universitarias de programación utilizando el lenguaje Gobstones. A cada uno se le asignaron 20 consignas (estímulos) al azar junto con los requerimientos didácticos que se utilizaron para generarlas. Además, para no interferir con la valoración de las consignas, los docentes no recibieron ninguna información acerca de cómo fueron generadas (es decir, desconocían sobre el uso de modelos de lenguaje para la generación) ni se hizo distinción alguna entre las consignas generadas por uno u otro tipo de *prompt*.

Para cada estímulo asignado los docentes analizaron y valoraron los siguientes aspectos: "¿La consigna dada está alineada con los requerimientos del docente?" (Escala Likert 1 a 5, donde 1 es Nada de acuerdo y 5 es Totalmente de acuerdo). "¿Cómo evaluás la calidad de la consigna en términos de la presencia de errores, ambigüedades y formulaciones confusas?", (Escala Likert 1 a 5, donde 1 es Muy baja calidad y 5 es Muy alta calidad). "Si tuvieses que evaluar los mismos requerimientos, ¿Utilizarías la consigna dada?", (Sí/No) "Acá podés justificar tus respuestas an-

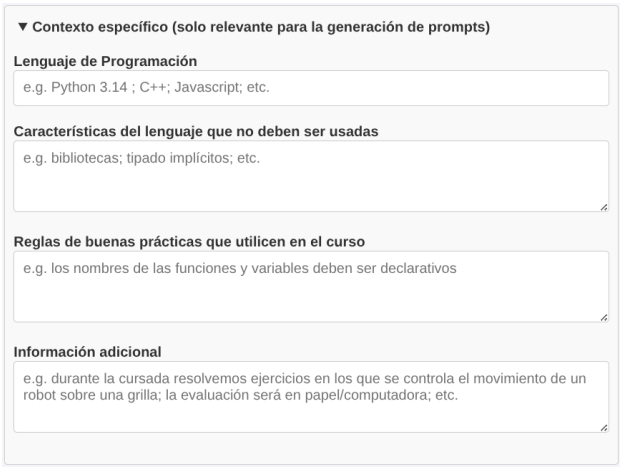

Figura 5. Pantalla principal de ProgrEval, campos opcionales

teriores o agregar cualquier comentario extra que te parezca pertinente al respecto.' (Campo de respuesta abierta)

4.3 Resultados

Las valoraciones promedio obtenidas para la pregunta *"¿La consigna dada está alineada con los requerimientos del docente?"* fueron 4,57 para los estímulos generados con *prompt* estructurado sin ontología, 4,13 para *prompt* estructurado con ontología y 4,34 para *prompt* sin estructura. En términos generales, los resultados muestran que los tres tipos de prompt utilizados obtuvieron valoraciones positivas en la escala utilizada (1: Nada de acuerdo, 5: Totalmente de acuerdo). Esto pone de manifiesto una alineación significativa de las consignas con los requerimientos especificados, destacándose aquellas generadas por *prompts* estructurados sin ontología, aunque contradice parcialmente nuestra hipótesis, siendo el prompt estructurado con ontología el que peor desempeño mostró.

Para responder a la segunda pregunta del trabajo "¿En qué medida las consignas generadas presentan una alta calidad?", analizamos la calidad de las consignas generadas a partir de las valoraciones docentes a la pregunta "¿Cómo evaluás la calidad de la consigna en términos de la presencia de errores, ambigüedades y formulaciones confusas?". Las consignas fueron valoradas positivamente para los prompts estructurados sin ontología (4,45) y sin estructura (3,85) en la escala utilizada (1: Muy baja calidad, 5: muy alta calidad). En esos casos, los docentes indicaron una baja presencia de errores, ambigüedades y formulaciones confusas. Acá observamos resultados similares a los de la pregunta anterior: las consignas generadas por *prompts* estructurados sin ontología fueron valoradas con mejor calidad (más de 0,6 puntos de diferencia del resto) mientras que aquellas generadas por *prompts* estructurados con ontología presentan la calidad más baja. Sin embargo, en este último caso es importante notar que una valoración alrededor de 3 indica una posición intermedia sobre la calidad de las consignas que vale la pena analizar con detalle a partir de los comentarios de los docentes.

Para obtener más evidencias acerca de la calidad de las consignas generadas, incorporamos al experimento la pregunta *"Si tuvieses que evaluar los mismos requerimientos, ¿Utilizarías la consigna dada?"*. Las respuestas posibles eran SÍ o NO, codificadas como 1 y 0, respectivamente. Frente a esta pregunta, los docentes consideraron que un poco más de la mitad de las consignas eran adecuadas para su uso en evaluaciones. Una vez más, se presenta una preferencia por las

consignas generadas con el *prompt* estructurado sin ontología, que alcanzaron un promedio de 0,74, mientras que el estructurado con ontología alcanzó 0,57 y el no estructurado 0,53.

Análisis de los comentarios Por último, analizamos comentarios que los docentes registraron para justificar su respuesta a las preguntas anteriores. Cabe aclarar que no todos los docentes participantes escribieron comentarios sobre todas las consignas evaluadas, y que hubo comentarios que no fueron asignados a una categoría (por falta de información o precisión). Por lo tanto, que una consigna no haya recibido comentarios sobre alguno de estos temas no implica que la crítica no le sea aplicable. Identificamos comentarios relacionados a:

- La alineación de la consigna con la propuesta didáctica del curso, por ejemplo, falta de conocimientos previos o nivel de dificultad diferente al esperado.
- Uso incorrecto de vocabulario técnico, por ejemplo, no distinguir entre conceptos como funciones y procedimientos, o utilizar diferentes denominaciones (repetición vs. directiva de repetición).
- Uso incorrecto del vocabulario asociado al dominio del problema, por ejemplo, hablar de bolitas y no de elementos del problema.
- Presencia de "malas prácticas" en el código provisto por la consigna, por ejemplo, usar denominaciones poco descriptivas o no abstraer aspectos del problema con funciones o procedimientos.
- Aplicación incorrecta de los formatos Parsons y Compleción de espacios vacíos, por ejemplo, ordenar fragmentos de código con 3 instrucciones o formulaciones ambiguas sobre los espacios a completar.
- Subespecificación del problema a resolver, por ejemplo, no se explicitan precondiciones de un problema o no se contemplan casos de borde en el código base.

Hipótesis de los resultados Los resultados nos indican que la propuesta está bien orientada pero requiere ajustes. Para empezar, la mera estructuración de los requerimientos probó mejorar sustancialmente la calidad de las consignas generadas, lo que era esperable ya que es sabido que los modelos dan mejores respuestas si el contexto proporcionado es más rico. Por otro lado, vemos un desempeño menor al esperado en las consignas generadas mediante *prompts* estructurados con ontología. La información adicional proporcionada por la ontología parece haber rebajado la calidad de las consignas, en algunos casos a niveles por debajo de aquellas generadas por los *prompts* sin estructura. Para dar explicación a esto planteamos las siguientes hipótesis que nos proponemos poner a prueba en trabajos futuros:

1. Tamaño de la ventana de contexto: dada la cantidad de información que se indica en el *prompt* estructurado con ontología, es posible que exceda la ventana de contexto de los modelos de lenguaje más básicos.
2. Popularidad del lenguaje: Gobstones es un lenguaje que está altamente subrepresentado en los sets de entrenamiento de los LLMs, es posible que no tenga suficientes ejemplos de referencia para generar nuevas consignas.
3. Expectativas de los docentes participantes: el criterio de los docentes participantes está fuertemente alineado con el método de enseñanza de Gobstones. Esto podría haber sesgado sus respuestas ya que, si bien los *prompts* indican el lenguaje a ser utilizado, no dicen nada acerca del modo en que se abordan los conceptos de programación y las estrategias didácticas que acompañan ese abordaje.
4. Falta de información en los prompts estructurados:
   (a) Calidad de los ejemplos de la ontología: la calidad y variedad de los ejemplos disponibles en la ontología tienen un amplio margen de mejora.
   (b) Definiciones de la ontología insuficientes:

   i. Los formatos Parsons y Compleción de espacios vacíos han sido los peor empleados por los LLMs. Definiciones más precisas de cada uno podrían mejorar el desempeño en este aspecto.
   ii. Las malas prácticas presentes en el código de algunas consignas son propias del concepto fundamental empleado y no del lenguaje de programación. Añadir esta información a la ontología podría guiar a los modelos para producir código de mejor calidad.

(c) Reglas de redacción y confección general de consignas en el prompt: ninguno de los prompts estructurados da indicaciones sobre qué caracteriza a una buena consigna de programación. Se podrían incluir lineamientos generales que guíen la redacción y eviten el uso incorrecto del vocabulario técnico o del dominio.

En cuanto al desempeño de las consignas generadas por uno u otro modelo, se podría pensar que hay una diferencia significativa en el desempeño de *Gemini* por sobre *GPT*. Sin embargo, dada la baja cantidad de datos recolectados, no podemos concluir cuantitativamente que se trata de un resultado representativo.

5 Conclusiones

Se mostró la propuesta completa de *ProgrEval*, una herramienta web que permite partir de un requerimiento de evaluación de un docente, hasta llegar a la generación de un *prompt* que el usuario puede materializar en una consigna usando su herramienta de IA generativa preferida.

Este trabajo incorpora el diseño de una ontología para la representación de evaluaciones de programación, que es utilizada para expandir el requerimiento de evaluación planteado por el usuario a través de la herramienta.

Finalmente, se observa que a través del uso de *ProgrEval* se pueden generar, manualmente o con la ayuda de una IA generativa, consignas de ejercicios que siguen los requerimientos de los usuarios. Aunque en este trabajo la información adicional provista por la ontología parece no tener un impacto significativo en la calidad y validez de las consignas generadasser aprovechada por los modelos utilizados en las pruebas, creemos que la herramienta puede acompañar a docentes en el proceso de confección, poniendo el foco en los requerimientos didácticos de la evaluación.

6 Trabajo futuro

ProgrEval no tiene suficientes ejemplos cargados para ser usada en un contexto general. Por consiguiente, además de seguir incorporando nuevas consignas, se desarrollarán nuevas formas en que los usuarios puedan colaborar sugiriendo consignas propias, que evaluaremos y clasificaremos manualmente.

Por otro lado, se buscará incluir la posibilidad de los usuarios puedan tener una versión propia de la ontología, agregando sus propios ejercicios a partir de una plantilla vacía, pudiendo luego exportar e importar a/desde un archivo local.

La interfaz de usuario aún tiene margen de mejora. Nos gustaría que las opciones de las distintas listas del formulario respondan a los contenidos que se encuentren realmente disponibles en la ontología, para que sea claro qué se puede obtener actualmente y qué no. En este sentido, se piensa incorporar un campo que permita especificar el dominio de la consigna que se desea generar (por ejemplo, un banco, una tienda de mascotas, una fábrica de automóviles, etc.) para añadir variabilidad a las consignas generadas.

Además, en lo inmediato volveremos a realizar una evaluación de la calidad de las consignas generadas, esta vez con el lenguaje Python, con el objetivo de ver cuánto de los resultados obtenidos se vio afectado por la utilización de un lenguaje poco común. Más adelante, llevaremos a cabo mejoras en los *prompts* estructurados para mitigar las otros puntos de falla identificados.

Por último, luego de completar esta última evaluación, se espera que la herramienta se comience a utilizar en cursos de programación universitarios y preuniversitarios, de forma de obtener *feedback* detallado de los docentes implicados.

Referencias

Atherton, J. S. (2011). Learning and Teaching; Bloom's Taxonomy.

Bormuth, J. R., & Menzel, P. (1970). On the theory of achievement test items.

Bower, M. (2008). A taxonomy of task types in computing. *Proceedings of the Conference on Innovation and technology in computer science education*, 281-285. https://doi.org/10.1145/1384271.1384346

Burke, C. M. (2025). AI-Assisted Exam Variant Generation: A Human-in-the-Loop Framework for Automatic Item Creation. *Education Sciences*, *15*(8). https://doi.org/10.3390/educsci15081029

Chan, K. W., Ali, F., Park, J., Sham, K. S. B., Tan, E. Y. T., Chong, F. W. C., Qian, K., & Sze, G. K. (2025). Automatic item generation in various STEM subjects using large language model prompting. *Computers and Education: Artificial Intelligence*, *8*, 100344. https://doi.org/10.1016/j.caeai.2024.100344

Doughty, J., Wan, Z., Bompelli, A., Qayum, J., Wang, T., Zhang, J., Zheng, Y., Doyle, A., Sridhar, P., Agarwal, A., Bogart, C., Keylor, E., Kultur, C., Savelka, J., & Sakr, M. (2024). A Comparative Study of AI-Generated (GPT-4) and Human-crafted MCQs in Programming Education. *Australian Computing Education Conference*, 114-123. https://doi.org/10.1145/3636243.3636256

Gierl, M., & Lai, H. (2016). Automatic Item Generation. Routledge/Taylor & Francis Group. https://doi.org/10.4018/978-1-5225-7365-4.ch016

Grover, S. (2020). Designing an Assessment for Introductory Programming Concepts in Middle School Computer Science. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 678-684. https://doi.org/10.1145/3328778.3366896

Khoirul Anam, R. (2025). Prompt Engineering and the Effectiveness of Large Language Models in Enhancing Human Productivity. https://doi.org/10.31219/osf.io/ad9y5_v1

Omopekunola, M. O., & Kardanova, E. Y. (2024). Automatic generation of physics items with Large Language Models (LLMs). *REID (Research and Evaluation in Education)*, *10*(2), 168-185. https://doi.org/10.21831/reid.v10i2.76864

Ruf, A., Berges, M., & Hubwieser, P. (2015). Classification of Programming Tasks According to Required Skills and Knowledge Representation. En V. J. Brodnik A (Ed.), *Informatics in Schools. Curricula, Competences, and Competitions* (pp. 57-68). Heidelberg: Springer.

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. *ACM Conference on International Computing Education Research*, 27-43. https://doi.org/10.1145/3501385.3543957

Simões, A., & Queirós, R. (2020). On the Nature of Programming Exercises. *arXiv preprint*. https://doi.org/10.48550/arXiv.2006.14476

Song, Y., Du, J., & Zheng, Q. (2025). Automatic item generation for educational assessments: a systematic literature review. *Interactive Learning Environments*, *33*, 1-20. https://doi.org/10.1080/10494820.2025.2482588

Tew, A. E., & Guzdial, M. (2010). Developing a validated assessment of fundamental CS1 concepts. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 97-101. https://doi.org/10.1145/1734263.1734297