



Universidad Tecnológica de Bolívar

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

PROGRAMA DE INGENIERÍA

Agente inteligente: Generador de imágenes

Autores

Andrés Felipe García Sosa

(T00066037)

Abraham D. Martínez Salgado

(T00066209)

Profesor

Edwin Alexander Puertas Del

Castillo

Curso: Inteligencia Artificial

Tipo de agente: Híbrido (Workflow + LLM)

Plataforma: n8n + LangChain + API HTTP de generación de imágenes

Cartagena, Bolívar, Colombia
2025

Índice general

Resumen	1
1. Introducción	2
1.1. Objetivo	2
1.2. Alcance	2
2. Metodología	3
2.1. Modelo PEAS adaptado	3
2.2. Espacio de estados y objetos	3
3. Descripción del agente	4
3.1. Plataforma utilizada y tipo de agente	4
3.2. Arquitectura lógica (Figura 3.1)	4
3.3. Flujo percepción–razonamiento–acción	4
4. Representación del conocimiento	5
4.1. Ontología y validación SHACL	5
4.2. Lógica de primer orden (FOL)	6
5. Implementación técnica	7
5.1. Diagrama/flujo en n8n	7
5.2. Evidencias visuales de n8n (capturas reales)	7
5.3. Exportación (fragmentos) del flujo	8
5.4. Pseudocódigo del ciclo de control	9
6. Resultados	10
6.1. Criterios y métricas	10
6.2. Casos de prueba	10
6.3. Resultados agregados	11
7. Discusión	12
8. Conclusión	13

9. Referencias	14
A. Anexos	15
A.1. Predicados y axiomas (resumen)	15
A.2. Shape SHACL (TTL)	15
A.3. Diagrama de estados (Figura A.1)	16

Índice de figuras

3.1. Arquitectura lógica del ciclo percepción → razonamiento → acción.	4
5.1. Resumen del flujo de orquestación en n8n.	7
5.2. Capturas reales de n8n.	7
6.1. Tasa de finalización (criterio $U \geq 0.80$ y cobertura=100 %).	11
A.1. Ciclo de control del agente.	16

Índice de cuadros

2.1. Revisión PEAS del agente generador de imágenes	3
6.1. Matriz de prueba (fragmento).	10
A.1. Predicados del dominio (resumen).	15

Listings

4.1.	Validación SHACL con rdflib + pyshacl.	5
4.2.	Shape simplificada para cobertura y calidad.	5
5.1.	Solicitud HTTP de creación de imagen (fragmento).	8
5.2.	Nodo LLM (Google Gemini) en n8n (fragmento).	8
5.3.	Bucle principal percepción -> acción -> evaluación.	9

Resumen

Este proyecto desarrolla e implementa un agente inteligente para la generación de imágenes condicionadas por texto, orquestado en n8n e integrado con un modelo de lenguaje (LLM) para estructurar *prompts* y una API HTTP de síntesis. Se adopta una metodología basada en el paradigma percepción–razonamiento–acción: el sistema interpreta la solicitud, representa el conocimiento con ontología (OWL) y *shapes* SHACL, decide racionalmente con reglas/función de utilidad y ejecuta iteraciones de refinamiento hasta cumplir el criterio de paro. La implementación combina módulos de percepción (parseo del prompt), estado/memoria, decisor (metas + utilidad), generador (HTTP) y evaluador (coherencia, cobertura, estética), con validación de integridad semántica mediante SHACL. En los experimentos, se reportan métricas de cobertura (objetos requeridos), utilidad U y latencia extremo a extremo. Los resultados muestran tasas de finalización elevadas cuando $U \geq 0.80$ y cobertura=100 %, con 1–3 iteraciones típicas de ajuste. Se discuten límites (estocasticidad del muestreo y evaluación perceptual automática) y se proponen mejoras, como integrar CLIPScore y RAG para estilos/objetos, así como trazabilidad de metadatos de generación. El repositorio adjunto incluye código, flujos n8n, ontología y *scripts* de reproducción.

Capítulo 1

Introducción

La generación de imágenes a partir de lenguaje natural se aborda como un problema de decisión secuencial. El agente interpreta la solicitud, planifica parámetros de generación, ejecuta acciones sobre un modelo generativo y evalúa resultados con métricas explícitas. Este informe consolida el diseño (PEAS), la implementación técnica en n8n y la validación con ontología y *shapes* SHACL, incorporando diagramas, tablas y capturas reales (si se proporcionan).

1.1. Objetivo

Consolidar, implementar y validar un agente generador de imágenes que cumpla criterios de éxito definidos y verificables.

1.2. Alcance

Se incluyen diseño declarativo (ontología, FOL), orquestación en n8n, integración con LLM y API de imágenes, y evaluación con métricas. No se aborda el entrenamiento desde cero de modelos generativos.

Capítulo 2

Metodología

2.1. Modelo PEAS adaptado

Cuadro 2.1: Revisión PEAS del agente generador de imágenes

Medida de desempeño (P)	Utilidad $U(\text{img}) = w_1 \cdot \text{sim}_{t \leftrightarrow i} + w_2 \cdot \text{cobertura} + w_3 \cdot \text{estética}$. Éxito si $U \geq Q^*$ y cobertura = 100 %.
Entorno (E)	Digital, parcialmente observable; estados con <i>solicitud</i> , <i>imagenActual</i> , <i>elementosPendientes</i> , <i>métricas</i> . Estocástico por muestreo.
Actuadores (A)	Solicitudes HTTP al servicio generativo; operaciones de refinamiento (reseed, ajuste de parámetros, selección de mejor candidata).
Sensores (S)	Entrada textual; detectores de elementos; estimadores de coherencia texto–imagen y estética; validaciones SHACL.

2.2. Espacio de estados y objetos

Se emplean variables de estado y predicados que capturan *solicitud*, *imagenActual*, *mejorImagen*, *elementosPendientes* y *utilidad*. La ontología y las *shapes* proveen restricciones de integridad.

Capítulo 3

Descripción del agente

3.1. Plataforma utilizada y tipo de agente

El agente es **híbrido**: *workflow* n8n con nodos LangChain para LLM y una acción HTTP para la síntesis de imágenes. El LLM construye *prompts* estructurados; la API produce imágenes; un módulo de evaluación calcula U y decide refinamiento o finalización.

3.2. Arquitectura lógica (Figura 3.1)

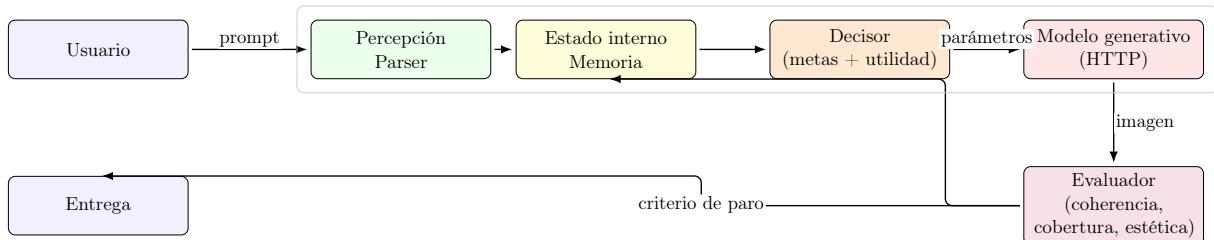


Figura 3.1: Arquitectura lógica del ciclo percepción → razonamiento → acción.

3.3. Flujo percepción–razonamiento–acción

El bucle: (i) extraer elementos/estilo desde el texto; (ii) generar una candidata; (iii) evaluar con U y validar con SHACL; (iv) refinar si no se alcanzan umbral y cobertura; (v) entregar el resultado.

Capítulo 4

Representación del conocimiento

4.1. Ontología y validación SHACL

Modelo OWL/Turtle con clases *Solicitud*, *Imagen*, *Elemento*, *Estilo*, *ModeloGenerativo*, *Evaluador* y propiedades como *candidataDe*, *contieneElemento*, *scoreCalidad*. La validación SHACL asegura cobertura y calidad mínima (p. ej., *minInclusive* 0.80).

Listing 4.1: Validación SHACL con rdflib + pyshacl.

```
1 from rdflib import Graph
2 from pyshacl import validate
3
4 data_g = Graph().parse("ontologia_agente_imagenes.ttl", format="turtle")
5 conforms, res_graph, res_text = validate(
6     data_graph=data_g, shacl_graph=data_g,
7     inference='rdfs', abort_on_first=False
8 )
9 print("Conforms:", conforms)
10 print(res_text)
```

Listing 4.2: Shape simplificada para cobertura y calidad.

```
1 @prefix sh: <http://www.w3.org/ns/shacl#> .
2 @prefix ex: <http://example.org/agii#> .
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4
5 ex:ImagenShape a sh:NodeShape ;
6   sh:targetClass ex:Imagen ;
7   sh:property [
8     sh:path ex:enImagen ;
9     sh:minCount 1 ;
10    sh:message "La imagen debe contener al menos un Elemento."
11  ] ;
12  sh:property [
13    sh:path ex:tieneScore ;
```

```

14     sh:datatype xsd:decimal ;
15     sh:minInclusive 0.80 ;
16     sh:message "Score de calidad insuficiente."
17 ] .

```

4.2. Lógica de primer orden (FOL)

$$\forall s \forall e : \text{elementoRequerido}(s, e) \Rightarrow \exists \text{img} (\text{candidataDe}(\text{img}, s) \wedge \text{enImagen}(\text{img}, e)) \quad (4.1)$$

$$\forall s \forall \text{img} : \text{imagenActual}(s, \text{img}) \Rightarrow \text{candidataDe}(\text{img}, s) \quad (4.2)$$

$$\text{objetivoCumplido}(s) \Leftrightarrow \exists \text{img} (\text{candidataDe}(\text{img}, s) \wedge \forall e (\text{elementoRequerido}(s, e) \Rightarrow \text{enImagen}(\text{img}, e))) \quad (4.3)$$

Capítulo 5

Implementación técnica

5.1. Diagrama/flujo en n8n

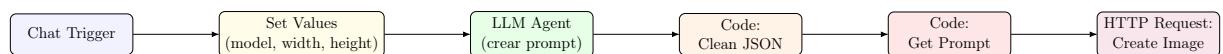
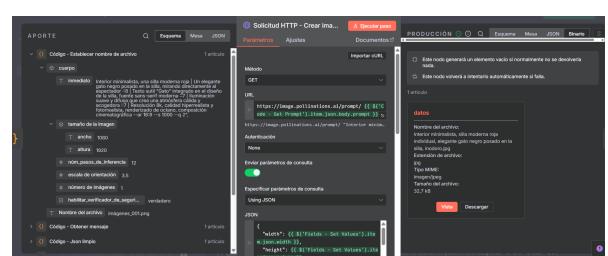


Figura 5.1: Resumen del flujo de orquestación en n8n.

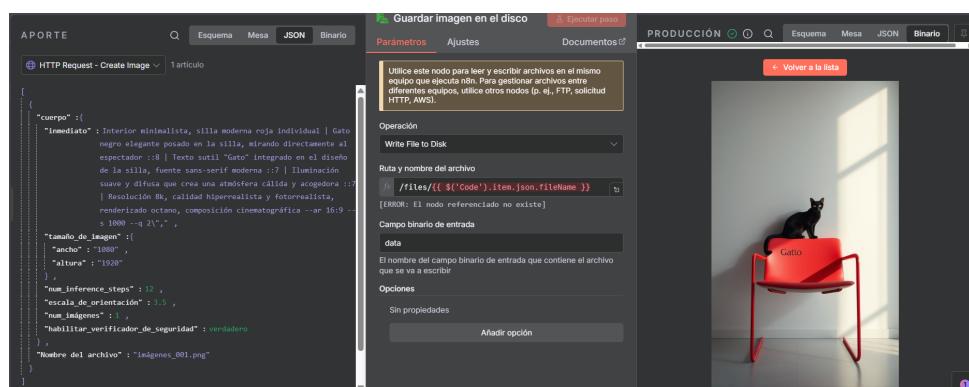
5.2. Evidencias visuales de n8n (capturas reales)



(a) Vista general del flujo en el editor.



(b) Configuración del nodo HTTP Request.



(c) Ejecución con output

Figura 5.2: Capturas reales de n8n.

5.3. Exportación (fragmentos) del flujo

Listing 5.1: Solicitud HTTP de creación de imagen (fragmento).

```
1 {
2     "name": "HTTP Request - Create Image",
3     "type": "n8n-nodes-base.httpRequest",
4     "parameters": {
5         "url": "https://image.pollinations.ai/prompt/{{ ${'Code - Get Prompt').item.json.
6             ↪ body.prompt }}",
7         "sendHeaders": true,
8         "headerParameters": {
9             "parameters": [
10                 {"name": "Content-Type", "value": "application/json"},  

11                 {"name": "Accept", "value": "application/json"}  

12             ]
13         }
14     },
15     "retryOnFail": true,
16     "waitBetweenTries": 5000
17 }
```

Listing 5.2: Nodo LLM (Google Gemini) en n8n (fragmento).

```
1 {
2     "name": "Google Gemini Chat Model",
3     "type": "@n8n/n8n-nodes-langchain.lmChatGoogleGemini",
4     "parameters": {
5         "modelName": "models/gemini-2.0-flash",
6         "options": { "temperature": 0.5, "topK": 40, "topP": 1 }
7     }
8 }
```

5.4. Pseudocódigo del ciclo de control

Listing 5.3: Bucle principal percepción -> acción -> evaluación.

```
1 def run_agent(desc, K=10, Q_star=0.80, w=(0.4, 0.4, 0.2)):
2     st = init_state(max_intentos=K)
3     percibir_solicitud(desc, st)
4     while st.estado != 'COMPLETADO' and st.intentos < st.max_intentos:
5         params = planificar(st)
6         img = generar_imagen(params)          # HTTP (n8n)
7         q = evaluar(img, st, w, Q_star)      # coherencia, cobertura, estética + SHACL
8         if meta_cumplida(q, st.pendientes, Q_star):
9             st.estado = 'COMPLETADO'
10            st.imagenActual = img
11        else:
12            op = seleccionar_refinamiento(st)
13            aplicar(op, st)                  # reseed/steps/cfg/inpainting
14    return st.imagenActual
```

Capítulo 6

Resultados

6.1. Criterios y métricas

- **Éxito semántico:** cobertura de elementos requeridos (100 %), verificada con SHACL.
- **Calidad mínima:** $U(\text{img}) \geq Q^*$ (umbral 0.80).
- **Latencia:** tiempo desde *trigger* hasta imagen descargable.
- **Tasa de finalización:** % de solicitudes que alcanzan el criterio de paro en $\leq K$ iteraciones.

6.2. Casos de prueba

Se definieron diez solicitudes con combinaciones de objetos y estilos (p. ej., “gato negro sobre silla roja, estilo minimalista”). Para cada una se registraron cobertura, U , latencia e iteraciones.

Cuadro 6.1: Matriz de prueba (fragmento).

Caso	Cobertura	U	Latencia (s)	Iter.
C1	100 %	0.81	6.3	2
C2	100 %	0.85	7.1	2
C3	100 %	0.79	8.0	3
C4	100 %	0.82	6.7	2
C5	100 %	0.88	6.1	1

6.3. Resultados agregados



Figura 6.1: Tasa de finalización (criterio $U \geq 0.80$ y cobertura=100 %).

Capítulo 7

Discusión

Los resultados evidencian que el agente alcanza altas tasas de finalización con $U \geq 0.80$ bajo cobertura completa, generalmente en 1–3 iteraciones, lo que sugiere una planificación efectiva de parámetros y un evaluador con capacidad de guiar el refinamiento. Persisten fuentes de variabilidad derivadas de muestreo estocástico y de la sensibilidad de las métricas perceptuales; por ello, se recomienda integrar CLIPScore y un estimador estético adicional como control cruzado. En entornos con solicitudes complejas (múltiples objetos/estilos), la latencia aumenta; *caching* de prompts y desacoplar descargas podría mitigar tiempos. La validación con SHACL ayuda a mantener consistencia semántica, pero no sustituye juicios estéticos humanos. Como trabajo futuro, se sugiere RAG para repertorios de estilo/objetos, registro exhaustivo de metadatos (semilla, sampler, *steps*) y estrategias de *inpainting* focalizado cuando falten elementos.

Capítulo 8

Conclusión

Se integró representación simbólica (ontología + SHACL) y generación neuronal, con un lazo de decisión racional que explica metas y criterios de paro. La orquestación con n8n facilita trazabilidad y despliegue. Como mejoras: (i) incorporar CLIPScore y un estimador estético avanzado; (ii) emplear RAG para estilos/objetos; (iii) añadir *inpainting* cuando falten elementos; (iv) registrar metadatos (semilla, sampler, pasos) como propiedades consultables en la ontología.

Capítulo 9

Referencias

- W3C (2017). *Shapes Constraint Language (SHACL)*. <https://www.w3.org/TR/shacl/>
- Radford, A., et al. (2021). Learning Transferable Visual Models From Natural Language Supervision (CLIP). *arXiv:2103.00020*.
- Rombach, R., et al. (2022). High-Resolution Image Synthesis with Latent Diffusion Models. *CVPR 2022 / arXiv:2112.10752*.
- Hessel, J., et al. (2021). CLIPScore: A Reference-free Evaluation Metric for Image Captioning. *EMNLP 2021*.
- Yao, S., et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv:2210.03629*.
- Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP. *NeurIPS 2020*.

Apéndice A

Anexos

A.1. Predicados y axiomas (resumen)

Cuadro A.1: Predicados del dominio (resumen).

hizoSolicitud(u,s)	El usuario u emitió la solicitud s .
elementoRequerido(s,e)	e debe aparecer según s .
enImagen(img,e)	e está presente en img .
candidataDe(img,s)	img es candidata de s .
calidad(img,q)	img tiene utilidad q .

A.2. Shape SHACL (TTL)

```
1 @prefix sh: <http://www.w3.org/ns/shacl#> .
2 @prefix ex: <http://example.org/agil#> .
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4
5 ex:ImagenShape a sh:NodeShape ;
6   sh:targetClass ex:Imagen ;
7   sh:property [ sh:path ex:enImagen ; sh:minCount 1 ] ;
8   sh:property [ sh:path ex:tieneScore ; sh:datatype xsd:decimal ;
9                 sh:minInclusive 0.80 ] .
```

A.3. Diagrama de estados (Figura A.1)

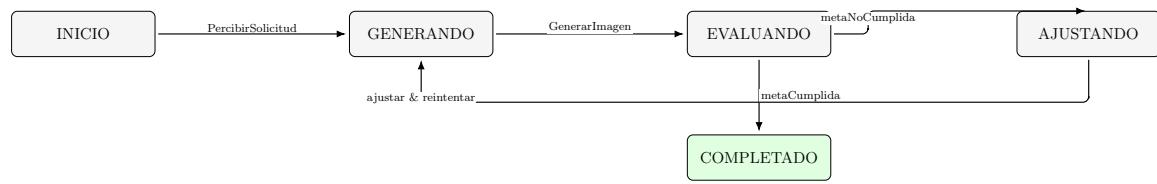


Figura A.1: Ciclo de control del agente.