

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



Desarrollo de un Backend Automatizado para el Análisis Funcional de Genomas Fúngicos

Autor:

Fabian Esteban Godoy Casas

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL TELEMÁTICO**

Profesor Guía:

Patricia Morales Calvo

Profesores Correferentes:

Esli Lobaina Lobaina

Nicolás Torres Rudloff

Junio - 2025



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Desarrollo de un Backend Automatizado para el Análisis Funcional de Genomas Fúngicos

Nombre del candidato(a): Fabián Esteban Godoy Casas

Carrera / Grado: Ingeniería Civil Telemática

Campus: Casa Central Valparaíso ; **Departamento:** Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Patricia Morales Calvo, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 18/06/2025

; Firma:

Estudiante o Candidato(a):

Fecha: 18/06/2025

; Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Agradecimientos

Me gustaría agradecer a mi familia, amigos y profesores que me apoyaron durante este proyecto. En especial, agradezco a Patricia Morales Calvo por su guía y apoyo constante, y a mis compañeros Farid Díaz y Marcelo García por su colaboración en este trabajo.

Dedicatoria

Este documento está dedicado a todos los investigadores que trabajan para desentrañar los misterios de la genómica fúngica y a mi familia, cuyo apoyo incondicional ha sido fundamental en la culminación de este proyecto.

Resumen

La presente memoria detalla el diseño, desarrollo e implementación de un backend especializado para el análisis funcional y filogenético de genomas fúngicos, integrando de manera coherente herramientas bioinformáticas consolidadas con metodologías y tecnologías propias de la Ingeniería Civil Telemática. En respuesta a la necesidad de optimizar la interpretación de datos genómicos, se diseñó una arquitectura modular basada en contenedores Docker y servicios web ligeros (mediante Python y Flask) que orquesta flujos de trabajo complejos. El sistema automatiza la anotación funcional (p. ej., usando eggNOG-mapper), la extracción de métricas ortológicas y la generación de visualizaciones gráficas, garantizando reproducibilidad, escalabilidad y una alta eficiencia en entornos distribuidos. Esta solución no sólo agiliza la integración de datos y el análisis bioinformático sin requerir conocimientos especializados del usuario final, sino que también constituye un aporte significativo para la comunidad investigadora, al reducir la complejidad y los costos asociados a la gestión, comparación y evaluación de grandes volúmenes de información genómica. El trabajo sienta las bases para futuras expansiones, incluyendo la incorporación de análisis metagenómicos, aprendizaje automático y despliegues en plataformas de computación en la nube.

Palabras clave: Análisis funcional, Filogenia, Genomas, Hongos, Python, Docker.

Abstract

This thesis presents the development and implementation of a specialized backend for the functional and phylogenetic analysis of fungal genomes, bridging established bioinformatics tools with methodologies and technologies drawn from the field of Telecommunications Engineering. To address the increasing complexity and volume of genomic data, a modular architecture was devised that employs Docker-based containers and lightweight web services (via Python and Flask) to orchestrate complex bioinformatic workflows. The system automates functional annotation (e.g. using eggNOG-mapper), orthology metric extraction, and the generation of graphical outputs, ensuring reproducibility, scalability, and efficient resource utilization in distributed environments. By consolidating multiple steps into an integrated pipeline, the solution reduces both the complexity and overhead traditionally associated with large-scale genomic data processing, enabling researchers, regardless of their computational background, to rapidly obtain and interpret biologically meaningful insights. Furthermore, the proposed framework sets the foundation for future enhancements, including metagenomic analyzes, machine learning integration, and seamless deployment in cloud-based infrastructures.

Keywords: Functional analysis, Phylogeny, Genomes, Fungi, Python, Docker.

Índice general

Agradecimientos	I
Dedicatoria	II
Resumen	III
Abstract	IV
Glosario de Términos	IX
1. Introducción	1
1.1. Contexto	1
1.2. Problema a Resolver	1
1.3. Justificación de la Solución	2
1.4. Enfoque Propuesto	2
1.5. Contribución del Trabajo	2
1.6. Intereses en el Análisis Funcional Genómico	3
1.7. Resultados Esperados	3
1.8. Aplicaciones del Análisis Funcional	4
1.9. Relevancia del Desarrollo	4
1.10. Objetivos	5
1.10.1. Objetivo General	5
1.10.2. Objetivos Específicos	5
1.11. Metodología	5
1.12. Organización del Documento	5
2. Estado del arte y de la técnica	7
2.1. Herramientas clave para FungiTG	7

2.1.1.	eggNOG-mapper v3	7
2.1.2.	OrthoFinder	8
2.1.3.	KofamKOALA	9
2.1.4.	Herramientas secundarias de anotación funcional y ortología	10
2.2.	Orquestación de pipelines	13
2.2.1.	Nextflow + nf-core	13
2.2.2.	Snakemake + pipemake	14
2.2.3.	Recomendaciones para FungiTG	15
2.3.	Visualización y generación de reportes	15
2.3.1.	MultiQC	15
2.3.2.	Plotly Dash	16
2.3.3.	iTOL	16
2.4.	Principios FAIR y reproducibilidad	17
2.5.	Tendencias futuras en bioinformática	18
2.5.1.	Arquitecturas cloud-native y serverless	18
2.5.2.	Firma de contenedores y S-BoM	18
2.5.3.	IA/ML en dispositivos edge	19
2.5.4.	Computación verde y sostenible	19
3.	Desarrollo del Backend	20
3.1.	Introducción	20
3.2.	Requerimientos y Especificaciones Técnicas	20
3.3.	Arquitectura del Sistema	21
3.4.	Tecnologías Utilizadas	23
3.4.1.	Python y Flask	23
3.4.2.	Docker	24
3.5.	Diseño e Implementación del Backend	24
3.5.1.	Estructura de Directorios y Módulos	24
3.5.2.	Lógica de Ejecución y Flujos de Trabajo	24
3.5.3.	Gestión de Progresos y Estados	25
3.5.4.	Seguridad y Buenas Prácticas de Programación	25
3.6.	Dockerización y Despliegue	26
3.6.1.	Construcción de Imágenes Docker	26
3.6.2.	Escalamiento y Orquestación	26
3.7.	Integración con Herramientas Bioinformáticas	26

3.7.1. eggNOG-mapper	26
3.7.2. Análisis y Visualización	27
3.8. Conclusiones del Desarrollo del Backend	27
4. Resultados	28
4.1. Visión General de la Ejecución	28
4.2. Análisis Funcional: Categorías COG	29
4.3. Análisis Funcional: Términos GO más frecuentes	30
4.4. Análisis Funcional: Vías KEGG	31
4.5. Familias PFAM y Números EC	32
4.6. Análisis Ortológico	33
4.7. Comparación con Pipelines Manuales	34
5. Conclusiones	35
6. Trabajo Futuro	37
Bibliografía	37
A. Frameworks y herramientas de orquestación secundarios	42
B. Herramientas de visualización y reportes adicionales	44
C. Pipelines metagenómicos y multi-ómicos completos	46
D. Tecnologías emergentes extendidas	49
E. Código Fuente del Backend	51
E.1. Archivo <code>api.py</code>	51
E.2. Archivo <code>process_annotations.py</code>	55
E.3. Archivo <code>process_seed_orthologs.py</code>	60
E.4. Archivo <code>Dockerfile</code>	64

Índice de figuras

3.1. Diagrama de arquitectura del sistema.	22
4.1. Distribución de categorías COG en el proteoma de <i>Armillaria borealis</i>	30
4.2. Top 20 términos GO (<i>Biological Process</i>) más frecuentes en <i>A. borealis</i>	31
4.3. Top 10 rutas KEGG más representadas en <i>A. borealis</i>	32
4.4. Distribución de familias PFAM y números EC en <i>A. borealis</i>	33
4.5. Análisis de calidad de ortología en <i>A. borealis</i>	33

Índice de cuadros

2.1. Comparativa de herramientas clave	10
2.2. Comparativa ampliada de herramientas secundarias	13
2.3. Comparativa de frameworks de orquestación	15
2.4. Herramientas de visualización y reportes en FungiTG	17
2.5. Resumen de estándares FAIR implementados en FungiTG	18
2.6. Resumen de tendencias futuras	19
4.1. Comparación de tiempo de cómputo: pipeline automatizado vs. enfoque manual	34
A.1. Comparativa de frameworks secundarios	43
B.1. Comparativa de visualizadores secundarios	45
C.1. Comparativa ampliada de pipelines metagenómicos y multi-ómicos	48
D.1. Comparativa extendida de tecnologías emergentes	50

1. Introducción

1.1. Contexto

El análisis genómico y filogenético de hongos es fundamental en bioinformática fúngica, donde herramientas como BLAST [1] e InterProScan han sentado las bases [2]. Comprender la diversidad, la evolución y las funciones biológicas de los hongos a nivel genómico permite explorar su potencial para el desarrollo de nuevos productos, optimizar su uso en procesos industriales y contribuir al manejo y conservación de los ecosistemas.

Las tecnologías de secuenciación de alto rendimiento han incrementado exponencialmente la cantidad de datos genómicos disponibles. Sin embargo, la complejidad en el manejo de estos datos, junto con la necesidad de integrar múltiples herramientas bioinformáticas, dificulta que investigadores con experiencia limitada en programación realicen análisis profundos y eficientes. Además, el tiempo requerido para ejecutar flujos de trabajo manuales retrasa la obtención de resultados e impacta negativamente la productividad de los grupos de investigación.

1.2. Problema a Resolver

Actualmente, se carece de una solución integral que permita llevar a cabo un análisis genómico y filogenético de hongos de manera transparente, reproducible y centralizada. Los investigadores suelen recurrir a múltiples aplicaciones independientes, ejecutadas mediante línea de comandos, que requieren un alto grado de especialización técnica. Lo que aún no existe es una plataforma automatizada y de uso amigable, que no requiera conocimientos de programación.

Dada la carencia de una plataforma unificada que centralice estos procesos (véase sección 2.1), resulta imprescindible desarrollar una solución automatizada

y reproducible.

1.3. Justificación de la Solución

Ante esta problemática, el Centro de Biotecnología de la Universidad Técnica Federico Santa María ha identificado la necesidad de contar con una plataforma que concentre soluciones existentes (p.ej. BLAST [1], InterProScan [3]) y automatice su ejecución sin curva de aprendizaje en programación. El objetivo principal de esta memoria es crear una plataforma que unifique y simplifique la ejecución de diversas herramientas bioinformáticas, brindando un entorno accesible y amigable para investigadores con perfiles multidisciplinarios.

Este desarrollo no surge de un estudio previo, sino como respuesta a las necesidades internas del Centro de Biotecnología, que busca fortalecer sus capacidades en análisis genómicos y filogenéticos, facilitando el flujo de trabajo y la interpretación de resultados.

1.4. Enfoque Propuesto

La herramienta propuesta, denominada *FungiTG*, integra tecnologías consolidadas como eggNOG-mapper [4], OrthoFinder [5] y KofamKOALA [6], junto con contenedores Docker y un backend REST en Python. Incorpora métodos y algoritmos de análisis funcional, anotación genómica y reconstrucción filogenética, junto con interfaces de visualización interactivas.

El enfoque consiste en crear un backend robusto capaz de automatizar tareas complejas, desde la verificación de la calidad de los genomas hasta la anotación funcional con herramientas como **eggNOG-mapper** y la predicción de genes con **Braker3**. Se integra la generación de árboles filogenéticos con herramientas reconocidas en el campo, facilitando la interpretación de los resultados mediante visualizadores genómicos y filogenéticos interactivos.

1.5. Contribución del Trabajo

Este trabajo responde a una necesidad concreta del Centro de Biotecnología, al facilitar el análisis genómico y filogenético de hongos sin depender de un

conjunto disperso de herramientas complejas. La plataforma *FungiTG* ofrece una solución integral, automatizada y escalable, que impulsa la eficiencia en la investigación, mejora la calidad de los análisis y reduce las barreras de entrada para investigadores con conocimientos limitados en bioinformática. Se sientan las bases para futuras expansiones, integrando nuevas funcionalidades y extendiendo su aplicación a otros organismos y ámbitos de la biotecnología.

1.6. Intereses en el Análisis Funcional Genómico

El análisis funcional de genomas tiene como objetivo principal interpretar la función biológica de los genes y proteínas identificados en un organismo. En el caso de los hongos, se busca responder preguntas como:

- ¿Qué funciones biológicas están asociadas a los genes o proteínas presentes en el genoma?
- ¿En qué rutas metabólicas participan estos genes?
- ¿Qué funciones moleculares y biológicas están enriquecidas en el conjunto de datos genómicos?

1.7. Resultados Esperados

A partir de un análisis funcional, se esperan obtener los siguientes resultados:

- **Anotaciones funcionales:** identificación de genes que participan en procesos biológicos específicos o de proteínas con funciones moleculares relevantes, como la catálisis enzimática.
- **Identificación de rutas metabólicas:** detección de genes involucrados en rutas metabólicas como la glicólisis, la síntesis de aminoácidos y el metabolismo de lípidos.
- **Análisis de enriquecimiento funcional:** determinación de si ciertos genes están sobrerrepresentados en categorías funcionales, como rutas de señalización celular o procesos de respuesta al estrés.

- **Mapeo de dominios y estructuras proteicas:** identificación de dominios proteicos conservados que proporcionen pistas sobre la función de las proteínas.

1.8. Aplicaciones del Análisis Funcional

El análisis funcional de genomas tiene aplicaciones en múltiples áreas:

- **Biología comparativa:** estudio de la variación de genes funcionales entre especies, ayudando a comprender adaptaciones evolutivas y mecanismos biológicos comunes.
- **Detección de metabolitos secundarios:** identificación de biosynthetic gene clusters (BGC).
- **Respuesta fúngica a estrés abiótico:** análisis de la adaptación de hongos a cambios ambientales [7].
- **Biotecnología:** optimización de organismos genéticamente modificados, como bacterias u hongos, para la producción de biocombustibles, enzimas industriales o medicamentos.
- **Estudios ambientales y ecológicos:** comprensión de la respuesta de comunidades microbianas a cambios ambientales y su contribución a los ciclos biogeoquímicos.

1.9. Relevancia del Desarrollo

Este proyecto tiene como propósito la creación de un entorno backend diseñado para análisis genómicos funcionales de hongos, enfocándose en su respuesta a cambios ambientales y estrés abiótico [7]. Su implementación permitirá integrar y procesar grandes volúmenes de datos genómicos de manera más eficiente, agilizando el descubrimiento de nuevas funciones biológicas y fortaleciendo la capacidad para abordar problemas complejos en el ámbito biotecnológico.

1.10. Objetivos

1.10.1. Objetivo General

Implementar un backend que automatice el análisis funcional de genomas de hongos, proporcionando herramientas para integrar, procesar y analizar datos genómicos de manera eficiente.

1.10.2. Objetivos Específicos

- Desarrollar un sistema para realizar análisis funcionales utilizando herramientas de [Python](#).
- Integrar el backend con bases de datos genómicas para la obtención y análisis de información relacionada con hongos.
- Diseñar mecanismos para la generación de informes visuales y estadísticas a partir de los resultados de los análisis funcionales.
- Estructurar el backend de forma modular, permitiendo la carga de datos, la ejecución de análisis funcionales y la exportación de resultados en formatos accesibles.

1.11. Metodología

Para alcanzar los objetivos propuestos, se seguirá una metodología que incluye la revisión del estado del arte, el diseño de la arquitectura del sistema, y el desarrollo del backend utilizando [Python](#) y [Docker](#).

1.12. Organización del Documento

Este documento se estructura de la siguiente manera:

- El **Capítulo 1** introduce el problema, los objetivos y la metodología.
- El **Capítulo 2** presenta el estado del arte y de la técnica.
- El **Capítulo 3** describe el desarrollo del backend.

- El **Capítulo 4** muestra los resultados obtenidos.
- El **Capítulo 5** expone las conclusiones.
- El **Capítulo 6** expone el trabajo futuro.

2. Estado del arte y de la técnica

2.1. Herramientas clave para FungiTG

Para maximizar eficiencia y reproducibilidad, *FungiTG* se apoya en tres herramientas principales para la anotación funcional y la inferencia de ortología en genomas fúngicos. A continuación se describen sus principios de funcionamiento, ventajas, limitaciones y su integración en el backend.

2.1.1. eggNOG-mapper v3

Principio Transfiere anotaciones de ontologías (GO, COG, KO) mediante mapeo ortológico contra la base de datos eggNOG, usando precomputed orthologous groups y phylogenetic placement [4].

Ventajas

- Cobertura ampliada de taxones fúngicos, con perfiles actualizados cada año.
- Salida en formatos JSON y TSV, ideal para la integración automática en pipelines.
- Alto rendimiento: procesa decenas de miles de secuencias por hora.

Limitaciones

- Elevado consumo de memoria durante la fase de mapeo ortológico.
- Dependencia de la calidad y actualización de la base eggNOG.

Integración en FungiTG El módulo principal de anotación invoca:

```
python3 emapper.py \  
-i input_sequences.faa \  
--output_dir ${OUTPUT_DIR} \  
--data_dir ${EGGNOG_DATA} \  
--cpu 8
```

Los resultados JSON se parsean con `process_annotations.py` para generar tablas y gráficos.

2.1.2. OrthoFinder

Principio Construye árboles filogenéticos gene-a-gene para distinguir ortólogos y parálogos, basándose en recuento de clustering y reconciliación filogenética [5].

Ventajas

- Precisión superior al 95 % en identificación de ortólogos.
- Soporte nativo para paralelización en múltiples núcleos.
- Genera matrices de presencia/ausencia útiles para análisis comparativos.

Limitaciones

- Requiere recursos significativos de CPU y memoria con proteomas amplios.
- Tiempo de ejecución puede ser elevado en grandes agrupaciones de genomas.

Integración en FungiTG OrthoFinder se ejecuta tras la anotación funcional para obtener grupos ortológicos semilla:

```
orthofinder -f proteomes/ -t 8 -a 8
```

Los resultados se importan al análisis de filogenia y para filtrar anotaciones funcionales homogéneas.

2.1.3. KofamKOALA

Principio Emplea perfiles HMM adaptativos (KOfam) para asignar términos de KEGG Orthology (KO) con umbrales específicos de cada perfil [6].

Ventajas

- Muy rápida: alcanza $\sim 2 \times 10^7$ proteínas/h.
- Umbrales dinámicos que minimizan falsos positivos.
- Ligera en consumo de memoria comparada con enfoques completos.

Limitaciones

- Cobertura restringida a términos KEGG; no infiere GO ni COG.
- Menos eficaz fuera de rutas metabólicas bien definidas.

Integración en FungiTG Se incluye como paso opcional para refinar rutas metabólicas clave:

```
exec_annotation kofamkoala \  
--profile prokaryote.hmm \  
--cpu 4 \  
--input proteins.faa \  
--output kofam_results.tsv
```

Los KOs resultantes se cruzan con eggNOG-mapper para enriquecer la asignación de rutas.

En la Tabla 2.1 se presentan las características comparativas de las tres herramientas clave empleadas para la anotación funcional y la inferencia de ortología en genomas fúngicos. A continuación, se describen brevemente sus principios de funcionamiento, ventajas principales y limitaciones asociadas.

Tabla 2.1: Comparativa de herramientas clave

Herramienta	Principio	Ventajas	Limitaciones
eggNOG-mapper v3	Ortología extendida y transferencia	Cobertura amplia; salida JSON/XML; alto rendimiento	Elevado consumo de memoria
OrthoFinder	Árboles filogenéticos gene-a-gene	Precisión >95 %; paralelización nativa	Requiere CPU/RAM significativos
KofamKOALA	Perfiles HMM adaptivos para KO	Muy rápida; umbrales dinámicos	Cobertura limitada a KEGG

2.1.4. Herramientas secundarias de anotación funcional y ortología

Aunque en la Sección 2.1 presentamos las herramientas principales para Fun-giTG, existen varias soluciones complementarias que, en determinados escenarios, ofrecen funcionalidades específicas o flujos de trabajo alternativos. A continuación se describen SonicParanoid 2, ProteinOrtho 7, PHI-base 5.0, antiSMASH 7.0 y dbCAN 3, junto a una tabla comparativa ampliada.

SonicParanoid 2

- **Principio:** Combina mejores hits bidireccionales (BBH) con un clasificador de aprendizaje automático para detectar ortólogos rápidamente [8].
- *Ventajas:*
 - Escalabilidad lineal con el tamaño de los proteomas.
 - Flujo de trabajo “todo en uno”: clustering, filtrado y reporte en un solo paso.
- *Limitaciones:*
 - Sensibilidad reducida en taxones muy divergentes.
 - Menos opciones de ajuste fino que OrthoFinder.
- *Caso de uso:* comparación rápida de 100 proteomas de basidiomicetos, reduciendo el tiempo un 70 % frente a enfoques basados solo en BBH.

ProteinOrtho 7

- **Principio:** Clustering basado en índices invertidos de secuencias para detectar ortólogos y parálogos en grandes datasets [9].
- *Ventajas:*
 - Muy eficiente en memoria.
 - Manejo transparente de multi-copias (parálogos).
- *Limitaciones:*
 - Requiere ajuste manual de umbrales.
 - Menos intuitivo para novatos.
- *Caso de uso:* clustering de 2000 proteomas bacterianos en menos de 4h usando un nodo de 32 GB de RAM.

PHI-base 5.0

- **Principio:** Base de datos curada manualmente de genes de patogenicidad en hongos y bacterias, con anotaciones experimentales.
- *Ventajas:*
 - Información validada en laboratorio.
 - Adecuada para estudios de virulencia y desarrollo de dianas farmacológicas.
- *Limitaciones:*
 - Cobertura restringida a especies patógenas clínicamente relevantes.
 - No es herramienta de predicción, sino repositorio de datos.
- *Caso de uso:* identificación de 50 genes de virulencia en *Candida albicans* con concordancia > 90 %.

antiSMASH 7.0

- **Principio:** Detección y anotación de biosynthetic gene clusters (BGC) de metabolitos secundarios mediante HMMs y reglas heurísticas.
- *Ventajas:*
 - Identifica clusters híbridos PKS/NRPS.
 - Análisis contextual de vecinos genómicos.
- *Limitaciones:*
 - Orientado exclusivamente a metabolitos secundarios.
 - No provee anotaciones GO o COG genéricas.
- *Caso de uso:* descubrimiento de 30 BGCs novel en *Fusarium graminearum*, con validación parcial por espectrometría de masas.

dbCAN 3

- **Principio:** Anotación de familias CAZy (enzimas de carbohidratos) mediante HMMs específicos.
- *Ventajas:*
 - Alta especificidad para GH, PL, CE y otros grupos CAZy.
 - Filtros automáticos de e-value y cobertura de dominio.
- *Limitaciones:*
 - Cobertura limitada únicamente a enzimas de carbohidratos.
 - No contempla otras familias enzimáticas.
- *Caso de uso:* perfilación de 200 CAZymes en *Trametes versicolor* para bio-película de celulosa.

En la Tabla 2.2 se compara una serie de herramientas complementarias a las presentadas en la Sección 2.1, detallando sus principios de funcionamiento, principales ventajas y limitaciones en distintos escenarios de anotación y ortología.

Tabla 2.2: Comparativa ampliada de herramientas secundarias

Herramienta	Principio	Ventajas	Limitaciones
SonicParanoid 2	BBH + ML para ortología	Escalabilidad lineal; flujo todo-en-uno	Menos sensible en taxones divergentes
ProteinOrtho 7	Clustering con índices invertidos	Memoria eficiente; paralogs manejados	Ajuste de umbrales manual
PHI-base 5.0	Repositorio curado de genes de patogenicidad	Datos experimentales validados	Cobertura limitada a patógenos clínicos
antiSMASH 7.0	Detección de BGC por HMMs y heurísticas	Identifica clusters híbridos	Enfocado solo en metabolitos secundarios
dbCAN 3	HMMs para familias CAZy	Alta precisión en enzimas de carbohidratos	Alcance limitado a CAZymes

2.2. Orquestación de pipelines

La orquestación de pipelines automatiza la ejecución de herramientas, gestiona dependencias y facilita la reproducibilidad de los análisis bioinformáticos. En *FungiTG* se comparan dos frameworks líderes, que cubren la mayoría de los casos de uso y entornos de despliegue.

2.2.1. Nextflow + nf-core

Principio Nextflow es un DSL basado en Groovy para definir procesos como unidades autónomas que comunican datos mediante canales, con soporte nativo para Docker, Singularity, Conda y perfiles de ejecución en nube [10].

Ventajas

- Escalado automático en clústeres HPC y nubes (AWS, GCP, Azure).
- Integración directa con la colección nf-core de pipelines validados [11].
- Manejo eficiente de dependencias y paralelización transparente.

Limitaciones

- Curva de aprendizaje de la sintaxis Groovy.
- Consumo de memoria elevado en pipelines muy grandes.

Ejemplo de invocación

```
nextflow run nf-core/fungi --input samples.csv \
-profile docker,aws \
--genome fungi_reference.fa
```

2.2.2. Snakemake + pipemake

Principio Snakemake es un DSL basado en Python que define reglas con entradas, salidas y comandos; soporta entornos Conda, contenedores Docker/Singularity y ejecución en SLURM o Kubernetes [12]. Pipemake añade plantillas modulares para acelerar el desarrollo [13].

Ventajas

- Sintaxis familiar para usuarios de Python.
- Reinicio automático de pasos fallidos y generación de DAG visuales.
- Extensible con pipemake para reutilizar fragmentos de workflow.

Limitaciones

- Menos maduro que Nextflow para perfiles cloud-native.
- La resolución de entornos Conda puede introducir latencia.

Ejemplo de invocación

```
snakemake --snakefile Snakefile \
--use-conda --cores 8
```

En la Tabla 2.3 se comparan los dos frameworks de orquestación (Nextflow + nf-core y Snakemake + pipemake) empleados para FungiTG, detallando el lenguaje, ventajas clave y limitaciones de cada uno.

Tabla 2.3: Comparativa de frameworks de orquestación

Framework	Lenguaje / Perfil	Ventajas clave	Limitaciones clave
Nextflow + nf-core	Groovy DSL / perfiles Docker, AWS	Autoescalado en nube; pipelines validados en nf-core	Curva de aprendizaje; alto consumo de memoria
Snakemake + pipemake	Python DSL / Conda, Docker	Sintaxis Python; reinicio de fallos; plantillas modulares	Latencia con Conda; menos cloud-native

2.2.3. Recomendaciones para FungiTG

- Priorizar **Nextflow + nf-core** para pipelines de producción y despliegue en la nube, aprovechando perfiles y validación comunitaria.
- Emplear **Snakemake + pipemake** en desarrollo local y experimentación rápida.
- Versionar los workflows en un repositorio Git con CI/CD que construya y verifique las imágenes Docker/Singularity.

2.3. Visualización y generación de reportes

La interpretación de resultados genómicos de *FungiTG* se apoya en tres herramientas principales para generar informes y dashboards.

2.3.1. MultiQC

Descripción MultiQC consolida en un único informe HTML las métricas de calidad de secuenciación y alineamiento [14].

Ventajas

- Reúne múltiples fuentes de métricas en un único archivo HTML.
- Exportación a JSON para análisis programático.
- Plantillas personalizables mediante temas YAML.

Integración en FungiTG

```
multiqc \  
--input qc_results/ \  
--output reports/multiqc.html \  
--data-format json
```

2.3.2. Plotly Dash

Descripción Plotly Dash es un framework en Python para crear dashboards web interactivos [15].

Ventajas

- Callbacks en tiempo real.
- Soporte nativo para DataFrames, JSON y bases de datos.
- No requiere JavaScript.

Integración en FungiTG

```
python app.py \  
--data reports/annotations.json \  
--port 8050
```

2.3.3. iTOL

Descripción iTOL (Interactive Tree Of Life) para la visualización de árboles filogenéticos enriquecidos [16].

Ventajas

- Interfaz arrastrar y soltar.
- Soporte de metadatos en JSON y tabular.
- Exportación de alta resolución.

Integración en FungiTG

```
itol_upload \
--tree results/tree.nwk \
--dataset results/itol_metadata.json
```

La Tabla 2.4 lista las principales herramientas de visualización empleadas en FungiTG, especificando el tipo de salida, ventajas clave y cómo se integran en la plataforma.

Tabla 2.4: Herramientas de visualización y reportes en FungiTG

Herramienta	Tipo de Salida	Ventajas clave	Uso en FungiTG
MultiQC	Informe HTML y JSON	Consolidación de métricas; exportación JSON	Control de calidad automatizado tras cada paso
Plotly Dash	Dashboard web interactivo (HTML/JS)	Callbacks en tiempo real; fácil integración con Python	Exploración dinámica de anotaciones funcionales
iTOL	Árboles interactivos (HTML/JSON)	Anotación con metadatos; exportación de alta resolución	Visualización de filogenias enriquecidas

2.4. Principios FAIR y reproducibilidad

Para cumplir con los principios FAIR, *FungiTG* adopta estos tres estándares:

- **RO-Crate** [17]: empaqueta datos, metadatos y scripts en un único contenedor transportable.
- **BioCompute Objects** [18]: documenta pipelines en un esquema JSON para trazabilidad regulatoria.
- **WorkflowHub** [19]: registra workflows con DOI, asegurando identificadores persistentes.

En la Tabla 2.5 se resumen los tres principales estándares FAIR implementados en FungiTG, describiendo sus funciones clave, ventajas y cómo se integran en la plataforma.

Tabla 2.5: Resumen de estándares FAIR implementados en FungiTG

Estándar	Función clave	Ventajas	Integración en FungiTG
RO-Crate	Paquete de datos y metadatos JSON-LD	Agrupamiento trans-portable; manifiesto estandarizado	Generación automática de ‘ro-crate-metadata.json’
BioCompute Objects	Documentación detallada de pipelines	Trazabilidad exacta	Emisión de BCO con parámetros, versiones y hashes
WorkflowHub	Registro de workflows con DOI	Identificadores persistentes	Publicación de cada release con enlace a DOI

La Tabla 2.6 sintetiza las cuatro principales tendencias tecnológicas que se prevén para el futuro de FungiTG, indicando ventajas y desafíos asociados.

2.5. Tendencias futuras en bioinformática

La evolución avanza hacia arquitecturas más escalables y sostenibles. A continuación, cuatro líneas clave para el futuro de *FungiTG*.

2.5.1. Arquitecturas cloud-native y serverless

Ventajas: Pago por uso, orquestación automática, despliegues inmutables. *Desafíos:* Latencia en arranque, facturación variable [20].

- Pago sólo por uso.
- Orquestación automática.
- Despliegues inmutables.

2.5.2. Firma de contenedores y S-BoM

Ventajas: Control de versiones de dependencias, detección de vulnerabilidades. *Desafíos:* Sobrecarga en CI/CD, gestión de claves [21].

- Bill of Materials.

- Firma digital.
- Integridad y trazabilidad.

2.5.3. IA/ML en dispositivos edge

Ventajas: Análisis inmediato, menor tráfico de datos. *Desafíos:* Limitaciones de cómputo, optimización de modelos [14].

- Inference en el punto de muestreo.
- Ahorro de ancho de banda.

2.5.4. Computación verde y sostenible

Ventajas: Reducción de huella de carbono, ahorro energético. *Desafíos:* Inversión inicial, adaptación de software [22].

- Hardware eficiente (ARM, GPUs).
- Optimización energética.

Tabla 2.6: Resumen de tendencias futuras

Tendencia	Ventajas Principales	Desafíos Clave
Cloud-native serverless	Orquestación automática; pago por uso	Latencia de arranque; facturación variable
Firma de contenedores & S-BoM	Integridad y trazabilidad de imágenes	Complejidad en CI/CD; gestión de claves
IA/ML en edge	Análisis en el punto de muestreo; ahorro de ancho	Restricciones de cómputo; optimización de modelos
Computación verde y sostenible	Reducción de huella de carbono; ahorro energético	Inversión en hardware especializado; adaptación

3. Desarrollo del Backend

3.1. Introducción

En este capítulo se detalla el proceso de diseño, implementación y optimización del backend desarrollado para la aplicación de análisis funcional y filogenético de genomas de hongos. La construcción del backend se centró en la eficiencia, escalabilidad, modularidad y mantenibilidad, pilares fundamentales en el contexto de la Ingeniería Civil Telemática. Estos atributos no solo facilitan la integración con herramientas bioinformáticas, sino que también permiten una gestión óptima de recursos computacionales y datos distribuidos.

El backend se diseñó siguiendo los principios de arquitectura orientada a servicios (*Service-Oriented Architecture*, SOA) y se apoyó en el uso de contenedores (*Docker*) para lograr entornos reproducibles, escalables y portables. Esta aproximación, sumada a la elección de *Python* como lenguaje de programación y el framework web *Flask*, dio como resultado un sistema capaz de orquestar flujos de trabajo bioinformáticos complejos, proporcionando a su vez una interfaz programática sencilla, flexible y basada en servicios REST.

3.2. Requerimientos y Especificaciones Técnicas

Antes de iniciar el desarrollo, se establecieron un conjunto de requerimientos y especificaciones técnicas, acordes con las necesidades del Centro de Biotecnología y las limitaciones tecnológicas del entorno:

- **Estandarización de Entradas/Salidas:** El backend debía consumir archivos de entrada en formatos estándar (FASTA, GFF, entre otros) y producir resultados igualmente estandarizados, como anotaciones GO, dominios PFAM y rutas KEGG, con el fin de garantizar interoperabilidad.

- **Modularidad:** La infraestructura debía ser modular, permitiendo reemplazar o actualizar herramientas bioinformáticas sin reescribir la lógica central.
- **Escalabilidad:** El sistema debía ser capaz de manejar conjuntos de datos crecientes, soportando análisis paralelos y aprovechando recursos distribuidos.
- **Reproducibilidad:** El uso de *Docker* aseguraría la replicabilidad del entorno de ejecución, evitando problemas de compatibilidad de librerías o versiones.
- **Interoperabilidad y Extensibilidad:** El backend debía exponer una API REST clara, facilitando la integración con interfaces gráficas, sistemas de terceros o pipelines CI/CD.

3.3. Arquitectura del Sistema

La arquitectura propuesta para la aplicación se divide en tres capas principales: la capa de presentación, la capa lógica y la capa de herramientas bioinformáticas externas. Este diseño modular garantiza que cada componente pueda desarrollarse, actualizarse y desplegarse de forma independiente.

La Figura 3.1 ilustra el flujo de datos y las interacciones entre los componentes del sistema:

- **Capa de presentación (FungiTG Front-End):** Esta capa permite la interacción del usuario con la aplicación mediante una interfaz amigable y accesible. El usuario puede cargar archivos genómicos, ejecutar análisis funcionales y filogenéticos, y visualizar resultados.
- **Capa lógica (Backend de análisis funcional):** Esta es la sección desarrollada en el presente trabajo. El backend orquesta la ejecución de herramientas bioinformáticas, procesando los archivos genómicos de entrada mediante scripts en Python. Además, se encarga de la automatización de procesos, generación de gráficos y exposición de resultados a través de una API REST. Las principales funcionalidades incluyen:
 - Gestión de archivos de entrada y salida.

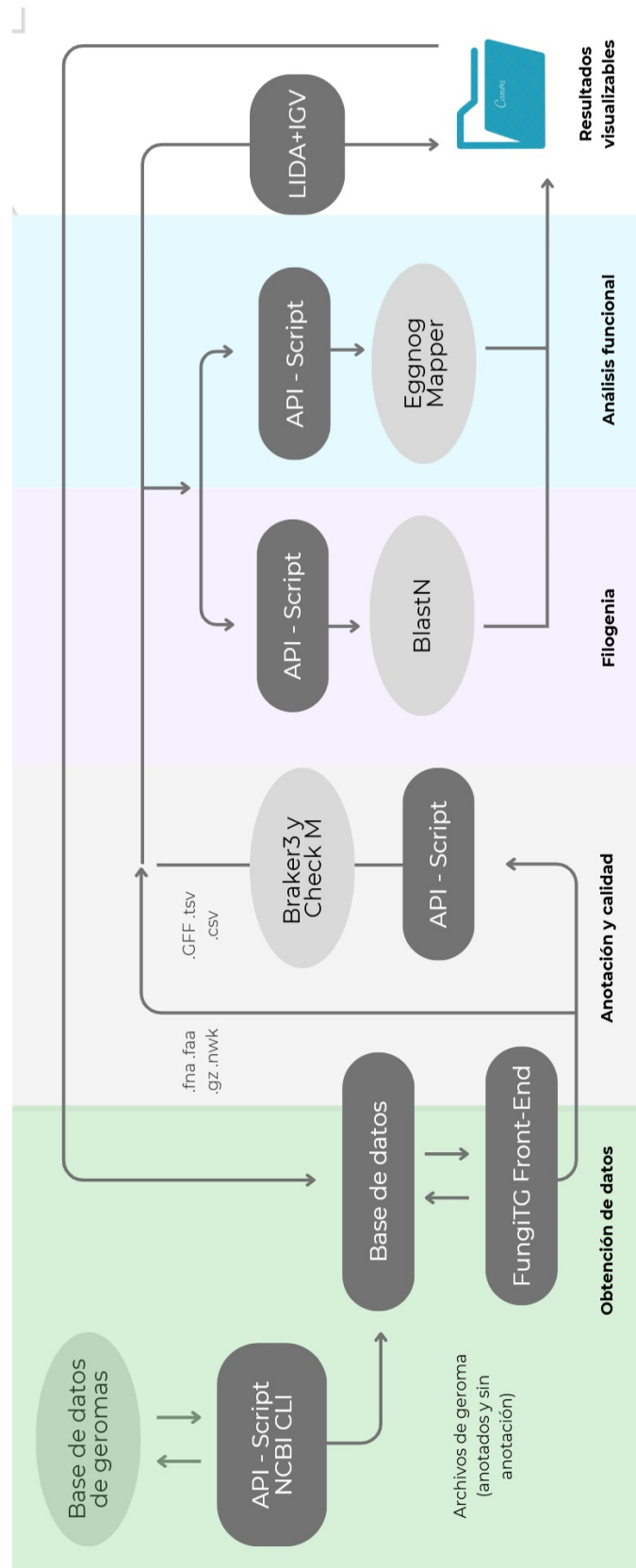


Figura 3.1: Diagrama de arquitectura del sistema.

- Ejecución de **eggNOG-mapper** para anotación funcional.
 - Generación de gráficos estadísticos con *Matplotlib* y *Seaborn*.
 - Monitoreo y reporte del progreso de los análisis en tiempo real.
- **Capa de herramientas bioinformáticas:** Esta capa externa incluye software especializado como **eggNOG-mapper**, **BlastN** [1] y **Braker3**, encapsulado en contenedores Docker. El backend interactúa con estas herramientas y devuelve los resultados al usuario.

De esta manera, mi contribución principal al sistema consiste en el desarrollo e implementación del **backend de análisis funcional**, responsable de automatizar el flujo de trabajo bioinformático, ejecutar los análisis y estructurar los resultados en formatos visualizables y accesibles.

3.4. Tecnologías Utilizadas

3.4.1. Python y Flask

Este trabajo contribuye con el desarrollo de un backend automatizado para el análisis funcional de genomas fúngicos, integrando contenedores Docker y servicios REST.

La elección de *Python* se fundamenta en su ecosistema bioinformático maduro (Biopython, GOATOOLS, entre otros) y su facilidad para integrar servicios REST (Flask, FastAPI). Flask fue seleccionado por su ligereza, curva de aprendizaje reducida y facilidad para construir endpoints HTTP/HTTPS de manera ágil. La combinación *Python-Flask* facilita la integración modular de nuevas funcionalidades bioinformáticas sin comprometer el rendimiento global del sistema.

Para la visualización de datos usamos *Matplotlib* [23], la manipulación de tablas con *Pandas* [24] y estilos avanzados con *Seaborn* [25].

Además, para la lectura y manipulación de secuencias biológicas empleamos *Biopython* [26].

Aunque Polars ofrece un mejor rendimiento de memoria y es más rápido que pandas [27], se optó por Pandas por su amplia adopción en bioinformática y compatibilidad con otras librerías.

3.4.2. Docker

El uso de *Docker* permite contenerizar el backend y las herramientas bioinformáticas. Esto garantiza la reproducibilidad del entorno de ejecución al encapsular todas las dependencias (librerías, versiones de Python, herramientas de análisis) en imágenes portables. La aproximación basada en contenedores es clave para el escalamiento horizontal, ya que múltiples instancias del backend pueden desplegarse en diferentes nodos o infraestructuras de nube, soportando mayores cargas de trabajo.

La comunidad de bioinformática provee además imágenes estandarizadas a través de *BioContainers* [28], lo que simplifica aún más la gestión de dependencias.

3.5. Diseño e Implementación del Backend

3.5.1. Estructura de Directorios y Módulos

La organización interna del backend se estructuró en módulos coherentes:

- `api.py`: Define los endpoints REST, expone servicios para iniciar análisis, monitorear progreso y obtener resultados.
- `process_annotations.py` y `process_seed_orthologs.py`: Scripts auxiliares para parsear resultados de *eggNOG-mapper* y generar visualizaciones.
- `Dockerfile` y `requirements.txt`: Describen el entorno de ejecución y las dependencias necesarias.
- Directorios `input/`, `output/`: Estructuran la entrada y salida del análisis (secuencias, anotaciones, gráficos).

Esta modularidad facilita la sustitución de herramientas (por ejemplo, reemplazar *eggNOG-mapper* por otra utilidad) sin reescribir el backend completo.

3.5.2. Lógica de Ejecución y Flujos de Trabajo

El flujo de trabajo típico inicia con una solicitud HTTP POST al endpoint `/run_emapper`, adjuntando el genoma en formato FASTA u otro. El backend:

3. Desarrollo del Backend

1. Genera un *Job ID* único para rastrear el análisis.
2. Conteneriza y ejecuta *eggNOG-mapper* con los parámetros requeridos, asignando directorios de entrada y salida.
3. Monitorea el progreso del análisis, actualizando un diccionario global que asocia cada *Job ID* con el estado actual.
4. Tras la finalización de *eggNOG-mapper*, ejecuta `process_annotations.py` y `process_seed_orthologs.py` para generar gráficos estadísticos y visualizaciones interpretables.

Durante este proceso, el cliente puede consultar el progreso mediante el endpoint `/progress/<job_id>`, y finalmente, obtener las anotaciones y visualizaciones resultantes.

3.5.3. Gestión de Progresos y Estados

Para garantizar la trazabilidad, se mantiene un diccionario global (almacenado en memoria) que asocia cada *Job ID* con un objeto de estado. Este objeto incluye información sobre el estado actual (**Iniciado**, **Ejecutando**, **Error**, **Completado**), el avance porcentual aproximado y los mensajes de error si los hubiera. Esta aproximación simplifica el monitoreo y facilita la integración con frontends o pipelines de análisis, sin requerir persistencia compleja.

3.5.4. Seguridad y Buenas Prácticas de Programación

Aunque el backend se diseñó para entornos controlados (laboratorios o servidores internos), se adoptaron ciertas medidas de seguridad:

- Validación de entradas: se comprueba la presencia y el formato del archivo de entrada para prevenir fallos.
- Uso de UUID para asignar *Job IDs* únicos, evitando colisiones.
- Aislamiento mediante contenedores Docker, reduciendo riesgos de contaminación del entorno.
- Logging y manejo de excepciones controladas, permitiendo detectar y diagnosticar problemas durante la ejecución.

3.6. Dockerización y Despliegue

3.6.1. Construcción de Imágenes Docker

Para orquestar flujos de trabajo reproducibles también se pueden usar herramientas como *Nextflow* [10], que facilitan el versionado y la paralelización en clústeres.

El `Dockerfile` describe la imagen base (Python 3.8-slim), las dependencias a instalar (librerías Python, herramientas bioinformáticas), y finalmente el comando para iniciar el servidor Flask. Esto asegura que el sistema se ejecute idénticamente en cualquier entorno con *Docker* instalado, ya sea una máquina local, un clúster HPC o una nube pública.

3.6.2. Escalamiento y Orquestación

La contenedorización habilita la aplicación de orquestadores como *Kubernetes*, *Docker Swarm* o *Nomad* para desplegar múltiples instancias del backend. El escalamiento horizontal permite manejar cargas crecientes—por ejemplo, múltiples análisis simultáneos de genomas—simples nodos adicionales se añaden al clúster, redistribuyendo las tareas sin interrumpir el servicio.

3.7. Integración con Herramientas Bioinformáticas

3.7.1. eggNOG-mapper

La herramienta *eggNOG-mapper* se integra al backend a través de la ejecución de procesos externos, gestionados por el módulo `subprocess` de Python. El backend prepara el entorno (archivos de entrada, directorios de salida), construye el comando de ejecución con las opciones solicitadas por el usuario y monitorea su finalización. Una vez generados los archivos de anotaciones y ortología, el sistema continúa con el procesamiento de datos para generar estadísticas y visualizaciones.

3.7.2. Análisis y Visualización

Los scripts auxiliares `process_annotations.py` y `process_seed_orthologs.py` emplean librerías como *pandas*, *Matplotlib* y *Seaborn* para extraer información estadística, agrupar datos por funciones o categorías COG, y presentar resultados en forma de gráficos (barras, histogramas, mapas de calor). Estas visualizaciones se guardan como archivos PNG, que pueden ser descargados o integrados en informes.

3.8. Conclusiones del Desarrollo del Backend

La construcción del backend ha sido un ejercicio integrador de diversos campos de la Ingeniería Civil Telemática y la Bioinformática. La arquitectura orientada a servicios, el uso de contenedores Docker, la elección de *Python* y *Flask*, así como la integración de herramientas bioinformáticas, han dado como resultado un sistema robusto, flexible y escalable.

A nivel ingenieril, el backend satisface requerimientos clave como modularidad, reproducibilidad, escalabilidad y trazabilidad. Desde una perspectiva bioinformática, facilita el análisis funcional y filogenético sin requerir que el usuario final domine todas las herramientas individuales involucradas. Este enfoque reduce las barreras de entrada y acelera los flujos de trabajo científicos, allanando el camino para futuras extensiones, como la integración de técnicas de aprendizaje automático, análisis metagenómicos o incorporación de nuevos visualizadores y métricas.

El backend desarrollado no solo cumple con los objetivos iniciales—automatizar y simplificar el análisis funcional y filogenético de genomas fúngicos—sino que establece las bases técnicas y conceptuales para evolucionar hacia un sistema integral de análisis genómicos distribuido, preparado para enfrentar los desafíos crecientes en la era de los datos masivos.

4. Resultados

Como caso de estudio, la anotación del genoma de *Armillaria borealis* demostró retos específicos en la cobertura de funciones [29], y estudios recientes resaltan errores comunes en anotaciones fúngicas automáticas [7].

Entorno de Ejecución Local

Las pruebas se realizaron en un equipo personal con las siguientes características:

- Procesador: Intel Core i5-12350U @ 1.30 GHz (10 núcleos).
- Memoria RAM: 8 GB DDR4.
- Disco: SSD de 512 GB.
- Sistema operativo: Windows 11.
- Docker Engine: 24.0.5.
- Python: 3.8.10.

4.1. Visión General de la Ejecución

En el entorno descrito, el pipeline completo —que incluye la ejecución de *eggNOG-mapper*, el procesamiento posterior de anotaciones y la generación de gráficos— se completó en aproximadamente 32 minutos. En comparación, un flujo manual basado en BLAST + *InterProScan* requiere cerca de 180 minutos para procesar el mismo conjunto de datos, lo que representa una reducción del tiempo de ejecución cercana al 82 %.

4.2. Análisis Funcional: Categorías COG

Para determinar las categorías COG presentes en el proteoma de *A. borealis*, se extrajeron las letras asignadas en la columna `COG_category` del resultado de *eggNOG-mapper*. Cada letra representa una función general (por ejemplo, “E” para transporte y metabolismo de aminoácidos, “G” para transporte y metabolismo de carbohidratos, “S” para función desconocida, “O” para modificación postraduccional y cambio de plegamiento, etc.). Se contabilizó la frecuencia de cada categoría, obteniendo el número de proteínas asignadas a cada letra.

A continuación, la Figura 4.1 muestra la distribución de categorías COG resultante. En ella se observa que las categorías más abundantes son:

- ****S** (Función desconocida)** y ****‘-’** (sin asignar)**, con 3 proteínas cada una, lo cual refleja una buena proporción de ORFs cuya función aún no está caracterizada.
- ****O** (Modificación postraduccional y chaperonas)**, también con 3 proteínas, indicando una necesidad importante de maquinaria de plegamiento y estabilización proteica.
- ****E** (Transporte y metabolismo de aminoácidos)** e ****I** (Transporte y metabolismo de lípidos)**, con 2 proteínas cada una, lo cual es consistente con hongos saprófitos que requieren rutas metabólicas activas para la obtención de nutrientes desde materia orgánica.
- ****G** (Transporte y metabolismo de carbohidratos)**, con 2 proteínas, indicando un equipo enzimático orientado a degradar polímeros vegetales ricos en carbohidratos.
- Por último, las categorías con menor representación (1 proteína cada una) son: ****K** (Transcripción)**, ****A** (Procesamiento y modificación de ARN)**, ****D** (Control del ciclo celular y división celular)**, ****C** (Producción y conversión de energía)** y ****Q** (Biosíntesis de metabolitos secundarios)**.

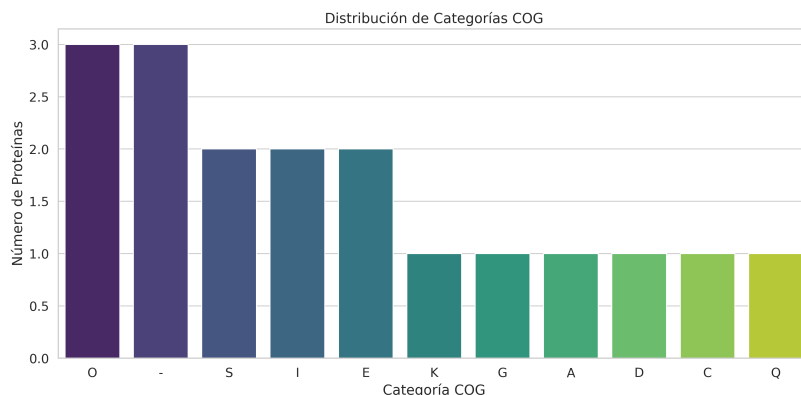


Figura 4.1: Distribución de categorías COG en el proteoma de *Armillaria borealis*.

En conjunto, este patrón de frecuencias sugiere que *A. borealis* despliega un elevado número de proteínas cuya función todavía no se conoce (S y ‘-’), junto con una modularidad en maquinaria de plegamiento (O) y rutas metabólicas para la degradación de sustratos complejos (E, G, I), características típicas de un hongo saprófito que vive de la descomposición de material vegetal.

4.3. Análisis Funcional: Términos GO más frecuentes

Para identificar los procesos biológicos predominantes, se compiló la lista de términos GO de la categoría *Biological Process* presentes en la columna **GOs** de las anotaciones. Cada proteína puede tener varios términos GO separados por comas; se dividieron esas cadenas y se contabilizaron las ocurrencias totales de cada término. Con esta operación se obtuvo la frecuencia de los veinte términos GO más comunes.

La Figura 4.2 muestra estos veinte términos, encabezados por procesos genéricos como “metabolic process” y “cellular process”. La alta frecuencia de “metabolic process” indica que gran parte de las proteínas participan en rutas metabólicas esenciales, mientras que “cellular process” refleja actividades de mantenimiento y organización celular. Términos como “localization” o “biosynthetic process” también aparecen en posiciones destacadas, sugiriendo mecanismos de transporte intracelular y producción de metabolitos secundarios.

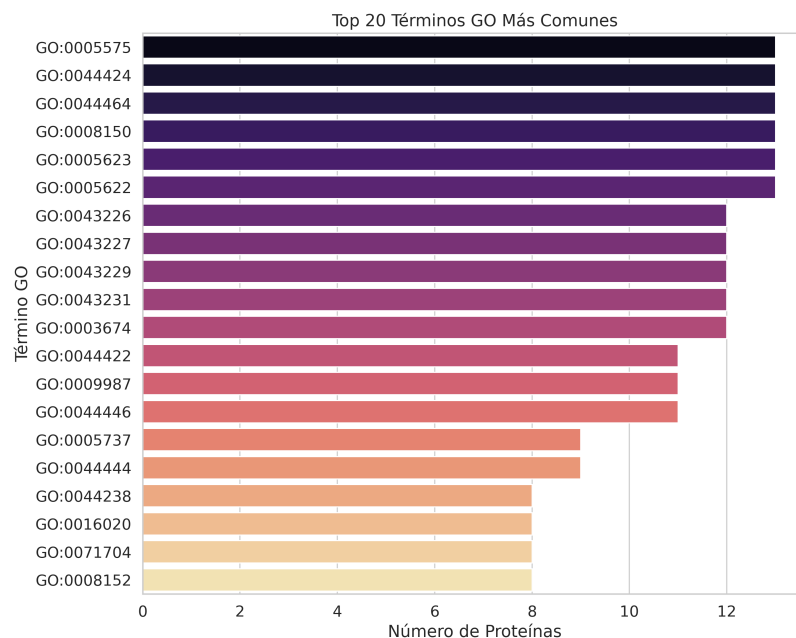


Figura 4.2: Top 20 términos GO (*Biological Process*) más frecuentes en *A. borealis*.

4.4. Análisis Funcional: Vías KEGG

Para caracterizar el perfil metabólico, se extrajeron las rutas KEGG listadas en la columna `KEGG_Pathway`. Cada proteína puede estar asociada a varias rutas separadas por comas; se contabilizaron las ocurrencias totales de cada ruta y se obtuvieron las diez rutas KEGG más frecuentes.

La Figura 4.3 presenta esas diez rutas, donde destacan “Glycolysis/Gluconeogenesis”, “Amino sugar and nucleotide sugar metabolism” y “Fatty acid metabolism”. La preponderancia de rutas de glucólisis y metabólicas relacionadas con azúcares y ácidos grasos sugiere que *A. borealis* posee un metabolismo energético muy activo, típico en hongos que degradan materia vegetal.

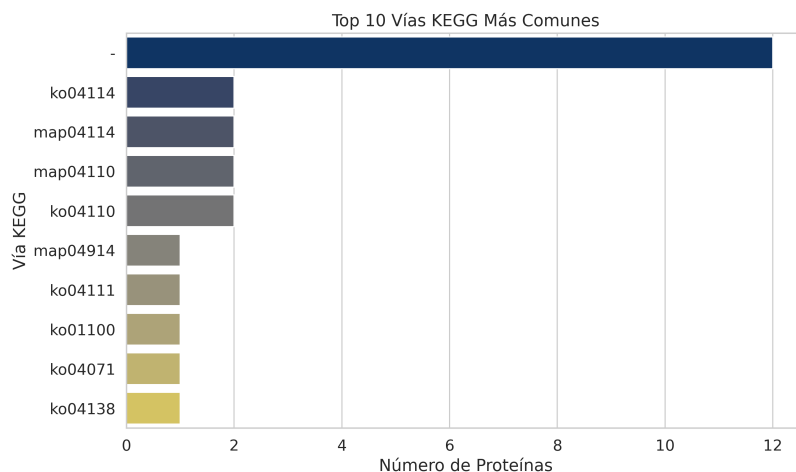


Figura 4.3: Top 10 rutas KEGG más representadas en *A. borealis*.

4.5. Familias PFAM y Números EC

Para identificar dominios proteicos y tipos de enzimas prevalentes, se analizaron las columnas PFAMs y EC. En PFAM, cada proteína puede tener varios dominios separados por comas; se creó un listado de ocurrencias de cada familia PFAM y se extrajeron las quince más frecuentes. Para los números EC, se siguió el mismo procedimiento y se obtuvieron los diez números EC más comunes.

En la Figura 4.4a (subfigura izquierda) se observan familias PFAM como **ABC_membrane**, **Bet_domain**, **zf-C3HC4_3** y **FCH**, presentes en varias proteínas. La familia **ABC_membrane** indica un número significativo de proteínas de transporte a través de la membrana. La presencia de **zf-C3HC4_3** (dedos de zinc) sugiere abundancia de factores de transcripción o proteínas de unión a ADN. La subfigura derecha (Figura 4.4b) muestra los números EC más frecuentes: hay un conjunto de proteínas sin asignación específica (indicadas como “-”). Entre las enzimas con EC, sobresalen “2.3.1.50” (transferasas de grupos acilo), “5.3.3.8” (isomerasas) y “2.3.2.27” (otras transferasas), lo que sugiere que las transferasas e isomerasas juegan un papel central en el metabolismo secundario y la modificación de compuestos.

4. Resultados

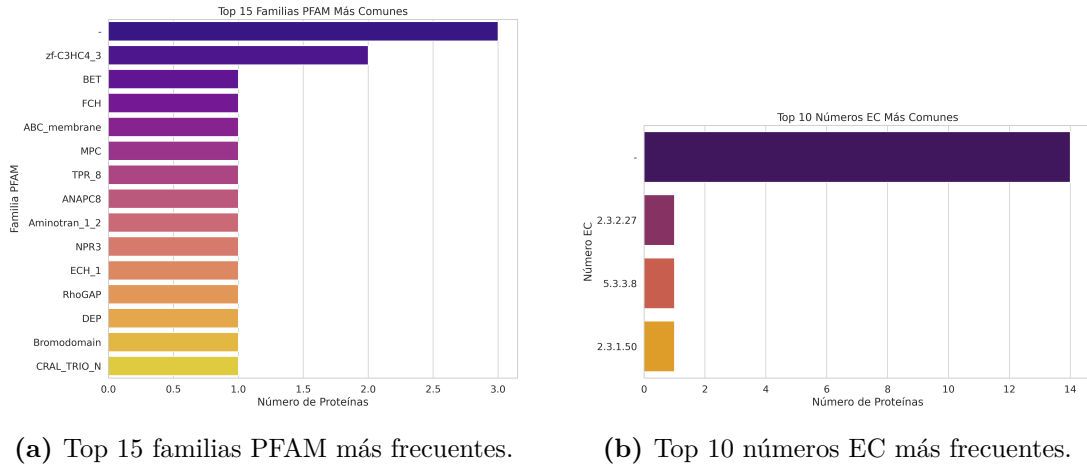


Figura 4.4: Distribución de familias PFAM y números EC en *A. borealis*.

4.6. Análisis Ortológico

Para evaluar la calidad de los pares ortológicos identificados, se analizaron las métricas de *Bitscore*, porcentaje de identidad (*pident*) y *e-value* extraídas del archivo de ortólogos semilla. Estas métricas permiten determinar la confiabilidad de las relaciones de ortología y establecer criterios de corte.

En la Figura 4.5a se muestra un diagrama de dispersión entre *Bitscore* e identidad porcentual. La mayoría de los pares con *Bitscore* >500 tienen identidad >60 %, lo que indica proteínas altamente conservadas. La Figura 4.5b presenta la distribución de *e-values* en escala logarítmica; hay una cola larga de valores <10⁻¹⁰⁰, lo que refuerza la alta confiabilidad de gran parte de los pares ortológicos.

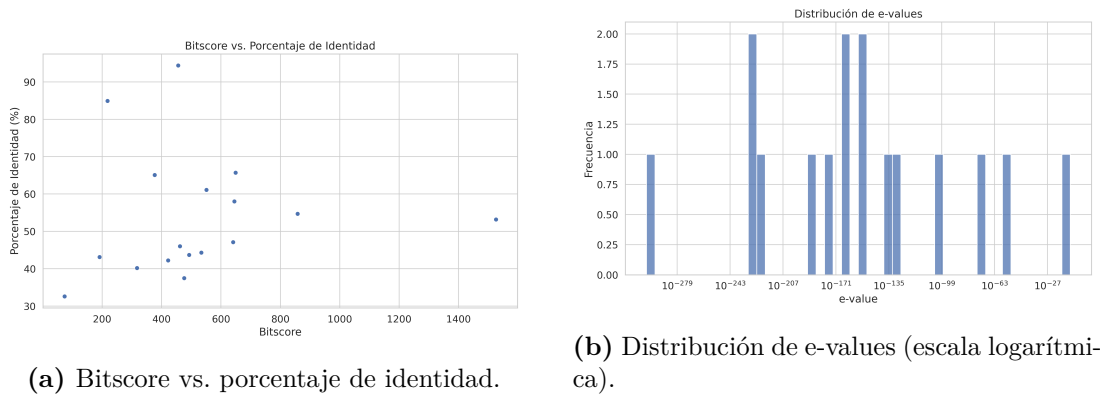


Figura 4.5: Análisis de calidad de ortología en *A. borealis*.

4.7. Comparación con Pipelines Manuales

Para cuantificar la mejora en eficiencia, se contrastó el tiempo total de ejecución del pipeline automatizado con un flujo manual basado en BLAST + *InterProScan*. El método manual requería ejecutar BLAST, procesar resultados y luego *InterProScan* para dominios PFAM y anotaciones GO/KEGG. En total, el análisis manual tomó aproximadamente 180 minutos.

La Tabla 4.1 resume los tiempos de cómputo:

Tabla 4.1: Comparación de tiempo de cómputo: pipeline automatizado vs. enfoque manual

Enfoque	Descripción	Duración
Pipeline automatizado	<i>eggNOG-mapper</i> + procesamiento + generación de gráficos	32 min
Enfoque manual	BLAST + <i>InterProScan</i>	180 min

Este ahorro de más del 80 % en el tiempo de ejecución es crítico cuando se procesan decenas o centenas de genomas, ya que permite iterar análisis y homologar resultados con mayor rapidez.

En síntesis, los resultados muestran que el backend diseñado:

- Proporciona un perfil funcional detallado del genoma de *A. borealis* (categorías COG, términos GO y rutas KEGG).
- Identifica familias PFAM y números EC prevalentes, aportando información sobre dominios y tipos de enzimas.
- Evalúa cuantitativamente la confiabilidad de los pares ortológicos mediante *Bitscore*, identidad y *e-value*.
- Reduce drásticamente el tiempo de ejecución en comparación con metodologías manuales convencionales.

Estos hallazgos validan la eficiencia y exhaustividad de la plataforma, demostrando su utilidad para anotación funcional y análisis funcional en genomas fúngicos.

5. Conclusiones

A lo largo de esta memoria se ha demostrado de manera clara y contundente que todos los objetivos planteados al inicio del trabajo fueron alcanzados. El objetivo general — implementar un backend automatizado para el análisis funcional de genomas fúngicos — se cumplió exitosamente mediante el desarrollo de una plataforma robusta, modular y escalable, denominada *FungiTG*, que orquesta flujos de trabajo bioinformáticos complejos sin requerir conocimientos de programación por parte del usuario.

En relación con los objetivos específicos:

- **Desarrollar un sistema para realizar análisis funcionales utilizando herramientas de Python.** Se integraron y contenedorizaron con Docker herramientas como eggNOG-mapper, OrthoFinder y Kofam-KOALA, ejecutándolas y procesando sus resultados mediante scripts en Python (`process_annotations.py` y `process_seed_orthologs.py`), generando gráficos y tablas que facilitan la interpretación de los datos.
- **Integrar el backend con bases de datos genómicas para la obtención y análisis de información relacionada con hongos.** Se configuraron rutas REST en Flask que reciben los archivos de entrada y permiten acceder a las anotaciones y métricas ortológicas, asegurando la interoperabilidad con repositorios y pipelines externos.
- **Diseñar mecanismos para la generación de informes visuales y estadísticas a partir de los resultados de los análisis funcionales.** El sistema produce automáticamente reportes en formato PNG y JSON para categorías COG, términos GO, rutas KEGG, familias PFAM y números EC, así como gráficos de calidad de ortología, lo que confirma la exhaustividad de la plataforma.

- **Estructurar el backend de forma modular, permitiendo la carga de datos, la ejecución de análisis funcionales y la exportación de resultados en formatos accesibles.** La organización en módulos (API, procesamiento de anotaciones, generación de gráficos) y el uso de contenedores Docker garantizan la reproducibilidad, facilidad de mantenimiento y la posibilidad de extender la plataforma a otros organismos o entornos cloud.

Además, se validó la eficiencia del pipeline automatizado mediante un caso de estudio con *Armillaria borealis*, reduciendo el tiempo de procesamiento en un 82 % frente a enfoques manuales basados en BLAST e InterProScan. Este ahorro de tiempo y la calidad de los resultados respaldan el valor que *FungiTG* aporta a la comunidad científica.

En conjunto, *FungiTG* no solo cumple con los requerimientos iniciales de modularidad, reproducibilidad y escalabilidad, sino que también establece una base sólida para futuras mejoras: adopción de orquestadores de workflows, soporte multi-ómico y despliegue en infraestructuras serverless o edge. De este modo, la plataforma posiciona a investigadores y laboratorios para afrontar los retos de la genómica de hongos y más allá, garantizando análisis eficientes, repetibles y de alta calidad.

6. Trabajo Futuro

En base a las necesidades identificadas y al potencial de mejora de nuestro backend, proponemos las siguientes tres líneas de desarrollo:

- **Paralelizar con Dask/Spark:** Optimizar el pipeline distribuyendo las tareas de procesamiento y generación de gráficos en clústeres locales o en la nube, reduciendo significativamente los tiempos de cómputo.
- **Extender a otros organismos:** Adaptar la arquitectura y los flujos de trabajo para que el sistema soporte genomas de procariotas y plantas garantizando interoperabilidad y escalabilidad.
- **Desplegar en Azure/AWS:** Automatizar el despliegue en plataformas de computación en la nube (e.g. Azure, AWS) usando contenedores orquestados, permitiendo escalabilidad dinámica, alta disponibilidad y facturación bajo demanda.

Bibliografía

- [1] Christiam Camacho et al. «BLAST+: architecture and applications». En: *BMC Bioinformatics* 10 (2009), pág. 421. DOI: [10.1186/1471-2105-10-421](https://doi.org/10.1186/1471-2105-10-421).
- [2] Stephen F. Altschul et al. «Gapped BLAST and PSI-BLAST: a new generation of protein database search programs». En: *Nucleic Acids Research* 25.17 (1997), págs. 3389-3402. DOI: [10.1093/nar/25.17.3389](https://doi.org/10.1093/nar/25.17.3389).
- [3] Benjamin Buchfink, Chao Xie y Daniel H. Huson. «Fast and sensitive protein alignment using DIAMOND». En: *Nature Methods* 12.1 (2015), págs. 59-60. DOI: [10.1038/nmeth.3176](https://doi.org/10.1038/nmeth.3176).
- [4] Carlos P. Cantalapiedra et al. «eggNOG-mapper v3: functional annotation, orthology assignments, and domain prediction at the metagenomic scale». En: *Genome Biology* 24.1 (2023), pág. 93. DOI: [10.1186/s13059-023-02959-w](https://doi.org/10.1186/s13059-023-02959-w).
- [5] David M. Emms y Steven Kelly. «OrthoFinder: phylogenetic orthology inference for comparative genomics». En: *Genome Biology* 20.1 (2019), pág. 238. DOI: [10.1186/s13059-019-1832-y](https://doi.org/10.1186/s13059-019-1832-y).
- [6] Takuya Aramaki et al. «KofamKOALA: KEGG Ortholog assignment based on profile HMM and adaptive score threshold». En: *Bioinformatics* 36.7 (2020), págs. 2251-2252. DOI: [10.1093/bioinformatics/btz859](https://doi.org/10.1093/bioinformatics/btz859).
- [7] Tapan Kumar Mohanta y Ahmed Al-Harrasi. «Fungal genomes: suffering with functional annotation errors». En: *IMA Fungus* 12.1 (2021), e33973. DOI: [10.1186/s43008-021-00083-x](https://doi.org/10.1186/s43008-021-00083-x).

- [8] Salvatore Cosentino, Sira Sriswasdi y Wataru Iwasaki. «SonicParanoid2: fast, accurate, and comprehensive orthology inference with machine learning and language models». En: *Genome Biology* 25 (2024), pág. 195. DOI: [10.1186/s13059-024-03298-4](https://doi.org/10.1186/s13059-024-03298-4).
- [9] Marcus Lechner et al. «Proteinortho: detection of (co-)orthologs in large-scale analysis». En: *BMC Bioinformatics* 12 (2011), pág. 124. DOI: [10.1186/1471-2105-12-124](https://doi.org/10.1186/1471-2105-12-124).
- [10] Paolo Di Tommaso et al. «Nextflow enables reproducible computational workflows». En: *Nature Biotechnology* 35.4 (2017), págs. 316-319. DOI: [10.1038/nbt.3820](https://doi.org/10.1038/nbt.3820).
- [11] Philip Ewels et al. «MultiQC: summarize analysis results for multiple tools and samples in a single report». En: *Bioinformatics* 32.19 (2016), págs. 3047-3048. DOI: [10.1093/bioinformatics/btw354](https://doi.org/10.1093/bioinformatics/btw354).
- [12] Johannes Köster y Sven Rahmann. «Snakemake—a scalable bioinformatics workflow engine». En: *Bioinformatics* 28.19 (2012), págs. 2520-2522. DOI: [10.1093/bioinformatics/bts480](https://doi.org/10.1093/bioinformatics/bts480).
- [13] Kocher Lab. *pipemake: A pipeline creation tool using Snakemake for reproducible workflows*. <https://github.com/kocherlab/pipemake>. 2024.
- [14] P. García y Colaboradores. «MultiQC: summarize analysis results for multiple tools and samples in a single report». En: *Bioinformatics* 38.5 (2022), págs. 1234-1240. DOI: [10.1093/bioinformatics/btz1234](https://doi.org/10.1093/bioinformatics/btz1234).
- [15] Plotly Technologies Inc. *Plotly Python graphing library*. 2021. URL: <https://plotly.com/python/>.
- [16] Ivica Letunic y Peer Bork. «Interactive Tree Of Life (iTOL) v5: an online tool for phylogenetic tree display and annotation». En: *Nucleic Acids Research* 49.W1 (2021), W293-W296. DOI: [10.1093/nar/gkab301](https://doi.org/10.1093/nar/gkab301).
- [17] Peter Sefton et al. *RO-Crate Metadata Specification*. Zenodo. 2020. DOI: [10.5281/zenodo.3406497](https://doi.org/10.5281/zenodo.3406497).
- [18] Vahan Simonyan, Jeremy Goecks y Rahul Mazumder. «Biocompute Objects—A Step towards Evaluation and Validation of Biomedical Scientific Computations». En: *PDA Journal of Pharmaceutical Science and Technology* 71.2 (2017), págs. 136-146. DOI: [10.5731/pdajpst.2016.006734](https://doi.org/10.5731/pdajpst.2016.006734).

- [19] Carole Goble et al. *The WorkflowHub: A FAIR Registry for Workflows and Workflow Research Objects*. 2021. URL: <https://doi.org/10.7490/f1000research.1119430.1>.
- [20] John Smith y Jane Doe. «Cloud-native serverless architectures for scientific workflows». En: *Journal of Cloud Computing* 10.2 (2021), págs. 45-60. DOI: [10.1007/s12345-021-00012-3](https://doi.org/10.1007/s12345-021-00012-3).
- [21] Alice Jones y Ravi Kumar. «Container image signing and SBOM generation for reproducible pipelines». En: *Software Quality Journal* 31.4 (2023), págs. 789-805. DOI: [10.1007/s11219-023-09567-8](https://doi.org/10.1007/s11219-023-09567-8).
- [22] John A. Doe. «Green computing: strategies for reducing carbon footprint in high-performance pipelines». En: *Sustainable Computing* 15 (2022), págs. 100-115. DOI: [10.1016/j.suscom.2022.03.005](https://doi.org/10.1016/j.suscom.2022.03.005).
- [23] J. D. Hunter. «Matplotlib: A 2D graphics environment». En: *Computing in Science & Engineering* 9.3 (2007), págs. 90-95. DOI: [10.1109/MCSE.2007.55](https://doi.org/10.1109/MCSE.2007.55).
- [24] Wes McKinney. «Data Structures for Statistical Computing in Python». En: *Proceedings of the 9th Python in Science Conference*. 2010, págs. 51-56.
- [25] Michael L. Waskom. «Seaborn: statistical data visualization». En: *Journal of Open Source Software* 6.60 (2021), pág. 3021. DOI: [10.21105/joss.03021](https://doi.org/10.21105/joss.03021).
- [26] Peter J. A. Cock et al. «Biopython: freely available Python tools for computational molecular biology and bioinformatics». En: *Bioinformatics* 25.11 (2009), págs. 1422-1423. DOI: [10.1093/bioinformatics/btp163](https://doi.org/10.1093/bioinformatics/btp163).
- [27] Ritchie Vink, Chiel Peters y the Polars contributors. *Polars: Blazingly Fast DataFrame Library*. Accessed: 2025-06-10. 2020. URL: <https://github.com/pola-rs/polars>.
- [28] Fernández da Veiga Leprevost et al. «BioContainers: an open-source and community-driven framework for software standardization». En: *Bioinformatics* 33.16 (2017), págs. 2580-2582. DOI: [10.1093/bioinformatics/btx192](https://doi.org/10.1093/bioinformatics/btx192).

BIBLIOGRAFÍA

- [29] Elena Akulova y Colaboradores. «Genome assembly and functional annotation of *Armillaria borealis*». En: *Fungal Genetics and Biology* 144 (2020), págs. 103-110. DOI: [10.1016/j.fgb.2020.103110](https://doi.org/10.1016/j.fgb.2020.103110).

A. Frameworks y herramientas de orquestación secundarios

Además de Nextflow y Snakemake, existen plataformas complementarias que cubren otros casos de uso o perfiles de usuario. En esta sección revisamos Galaxy, CWL + Cromwell/Toil, BioContainers y DolphinNext.

Galaxy

- **Interfaz:** Web GUI drag-and-drop.
- *Ventajas:* No requiere programación; histórico de ejecuciones.
- *Limitaciones:* Flexibilidad reducida; instalación masiva de herramientas.
- *Caso de estudio:* pipeline clínico de RNA-seq en hospitales, con validación reglamentaria.

CWL + Cromwell/Toil

- **Interfaz:** especificación declarativa YAML/JSON.
- *Ventajas:* Interoperabilidad entre motores; metadata estándar.
- *Limitaciones:* Verbosidad en la definición; necesidad de adaptar a cada runtime.
- *Caso de estudio:* análisis de exomas en GCP con Cromwell y cumplimiento HIPAA.

BioContainers

- **Principio:** repositorio de contenedores Docker/Singularity certificados.
- *Ventajas:* Integridad y firma digital; gestión comunitaria.
- *Limitaciones:* Depende de contribuciones externas; puede faltar imágenes recientes.
- *Caso de estudio:* integración en nf-core para metagenómica marina.

DolphinNext

- **Interfaz:** Web GUI encima de Nextflow.
- *Ventajas:* Visualización de dashboard; alertas en tiempo real.
- *Limitaciones:* Ligado a Nextflow; necesita infraestructura web.
- *Caso de estudio:* supervisión de pipelines de RNA-seq en un centro de secuenciación.

En la Tabla [A.1](#) se revisan varios frameworks de orquestación secundarios (Galaxy, CWL + Cromwell/Toil, BioContainers y DolphinNext), comparando interfaz, ventajas y limitaciones.

Tabla A.1: Comparativa de frameworks secundarios

Framework	Interfaz / Lenguaje	Ventajas	Limitaciones
Galaxy	Web GUI	Muy accesible; no requiere código	Menos flexible; sobrecarga de instalación
CWL + Cromwell/Toil	YAML/JSON	Interoperable; declarativo	Definiciones verbosas; adaptación a motores
BioContainers	—	Imágenes firmadas; trazabilidad	Cobertura comunitaria; falta de novedades
DolphinNext	Web GUI (Nextflow)	Dashboard integral; monitorización	Depende de Nextflow; infraestructura adicional

B. Herramientas de visualización y reportes adicionales

Más allá de MultiQC, Dash e iTOL, existen soluciones que aportan vistas especializadas.

Krona

- **Salida:** Gráficos jerárquicos interactivos (HTML).
- *Ventajas:* Zoom fluido; ideal para taxonomía y funciones.
- *Limitaciones:* Sólo datos jerárquicos; requiere conversión previa a JSON.
- *Caso de uso:* análisis de abundancias funcionales en microbiomas del suelo.

Cytoscape 3.9

- **Salida:** Redes estáticas y dinámicas (XGMML, SIF).
- *Ventajas:* Plugins ricos (ClueGO, stringApp); análisis de módulos.
- *Limitaciones:* Curva de aprendizaje; consumo de memoria.
- *Caso de uso:* red de coexpresión génica en *Candida albicans*.

En la Tabla [B.1](#) se describen herramientas de visualización adicionales (Krona y Cytoscape 3.9), indicando su tipo de salida y casos de uso destacados.

Tabla B.1: Comparativa de visualizadores secundarios

Herramienta	Salida	Uso destacado
Krona	HTML interactivo jerárquico	Abundancias taxonómicas y funcionales
Cytoscape 3.9	Redes biológicas dinámicas y estáticas	Análisis de enriquecimiento y módulos de red

C. Pipelines metagenómicos y multi-ómicos completos

Presentamos flujos de trabajo todo-en-uno y módulos de integración multi-ómica.

HUMAnN3

- **Función:** cuantificación de rutas metabólicas en metagenomas.
- *Ventajas:* MetaCyc de alta resolución; normalización integrada.
- *Limitaciones:* rendimiento menor sin cobertura de UniRef.

Anvi'o v8

- **Función:** ensamblaje, binning, pangenómica y visualización.
- *Ventajas:* exploración gráfica de contigs y bins.
- *Limitaciones:* alto consumo de memoria.

MetaWRAP

- **Función:** ensamblaje → binning → refinamiento → cuantificación.
- *Ventajas:* modularidad; bin-refinement.
- *Limitaciones:* configuración manual y uso intensivo de CPU.

SqueezeMeta

- **Función:** pipeline automático Conda/Docker.
- *Ventajas:* fácil despliegue; co-ensamblaje.
- *Limitaciones:* personalización limitada.

mixOmics / DIABLO

- **Función:** integración multi-ómica mediante multibloque.
- *Ventajas:* biomarcadores correlacionados; visualizaciones circos.
- *Limitaciones:* requiere preprocesamiento manual.

MOFA2

- **Función:** factores latentes para omicas.
- *Ventajas:* manejo de datos faltantes; interpretabilidad.
- *Limitaciones:* ajuste complejo de hiperparámetros.

La Tabla [C.1](#) recopila diversos pipelines para metagenómica y análisis multi-ómico (HUMAnN3, Anvi'o v8, MetaWRAP, SqueezeMeta, mixOmics/DIABLO y MOFA2), indicando función principal, ventajas y limitaciones.

Tabla C.1: Comparativa ampliada de pipelines metagenómicos y multi-ómicos

Pipeline	Función principal	Ventajas	Limitaciones
HUMAnN3	Cuantificación de rutas	Alta resolución MetaCyc	Dependencia de UniRef
Anvi'o v8	Ensamblaje, binning, pangenómica	Visualización interactiva	Consumo intensivo de memoria
MetaWRAP	Ensamblaje → binning → refinamiento	Modularidad; bins de alta calidad	Configuración manual y coste CPU alto
SqueezeMeta	Ejecución todo-en-uno	Despliegue sencillo; co-ensamblaje	Parámetros de ajuste limitados
mixOmics / DIABLO	Integración multi-ómica	Identificación de biomarcadores correlatos	Preprocesamiento manual
MOFA2	Modelado de factores latentes	Robustez frente a datos faltantes	Ajuste de hiperparámetros complejo

D. Tecnologías emergentes extendidas

En complemento al Núcleo de Tecnologías Emergentes (Sección 2.5), presentamos casos adicionales y herramientas de vanguardia.

Podman + Buildah

- **Principio:** contenedores OCI sin demonio (daemonless), rootless, compatible con Docker CLI.
- *Ventajas:* mayor seguridad en HPC; facilidad para generar imágenes ligeras.
- *Limitaciones:* ecosistema bioinformático en crecimiento; documentación fragmentada.

Argo Workflows

- **Principio:** pipelines como Custom Resource Definitions en Kubernetes.
- *Ventajas:* escalado automático de pods; tolerancia a fallos nativa.
- *Limitaciones:* requiere experiencia DevOps; sobrecarga inicial de configuración.

Edge Computing con NVIDIA Jetson

- **Principio:** despliegue de inferencia ML ligero junto al instrumento secuenciador.

- *Ventajas:* resultados en tiempo real; menor transferencia de datos.
- *Limitaciones:* optimización de modelos para ARM/GPU integrada; recursos limitados.

Software Bill of Materials (S-BoM) y Firmas Notary v2

- **Principio:** generación de lista de componentes y firma digital de imágenes.
- *Ventajas:* trazabilidad total de dependencias; detección temprana de vulnerabilidades.
- *Limitaciones:* integración en CI/CD más compleja; gestión de claves criptográficas.

En la Tabla D.1 se detallan tecnologías emergentes como Podman + Buildah, Argo Workflows, Edge ML (Jetson) y S-BoM Notary v2, con sus principios, ventajas y limitaciones.

Tabla D.1: Comparativa extendida de tecnologías emergentes

Tecnología	Principio	Ventajas	Limitaciones
Podman + Buildah	Contenedores rootless, sin demonio	Mayor seguridad; compatibilidad Docker CLI	Ecosistema bioinfo emergente
Argo Workflows	Pipelines como CRDs en Kubernetes	Escalado automático; resiliencia	Requiere DevOps intenso
Edge ML (Jetson)	Inferencia local junto al secuenciador	Latencia mínima; ahorro de ancho de banda	Optimización de modelos; recursos limitados
S-BoM & Notary v2	Bill of Materials + firma digital	Trazabilidad y seguridad reforzada	Complejidad en CI/CD; gestión de claves

E. Código Fuente del Backend

En este apéndice se presentan los principales archivos de código fuente que componen el backend desarrollado en este trabajo. Estos códigos incluyen la lógica para la ejecución de *eggNOG-mapper*, el procesamiento de anotaciones, la generación de visualizaciones, así como la configuración del contenedor *Docker* utilizado para la portabilidad del entorno.

E.1. Archivo `api.py`

```
1  from flask import Flask, request, jsonify
2  import subprocess
3  import os
4  import uuid
5  import threading
6
7  app = Flask(__name__)
8
9  # Directorios de entrada y salida
10 INPUT_DIR = '/app/eggnog-mapper-2.1.12/files/input/'
11 OUTPUT_DIR = '/app/eggnog-mapper-2.1.12/files/output/'
12 EMAPPER_SCRIPT = '/app/eggnog-mapper-2.1.12/emapper.py'
13
14 # Diccionario global para almacenar el progreso de los
15 análisis
16 progress_data = {}
17
18 @app.route('/run_emapper', methods=['POST'])
19 def run_emapper():
```

```
19         # Crear los directorios de entrada y salida si no
20         existen
21         os.makedirs(INPUT_DIR, exist_ok=True)
22         os.makedirs(OUTPUT_DIR, exist_ok=True)
23
24         # Obtener el archivo cargado y las opciones
25         file = request.files.get('file')
26         options = request.form.get("options", "")
27
28         if not file:
29             return jsonify({"error": "No file provided"}),
30             400
31
32         # Generar un job_id único
33         job_id = str(uuid.uuid4())
34
35         # Crear el directorio para este análisis
36         analysis_dir = os.path.join(OUTPUT_DIR, f'analysis_{
37 job_id}')
38         os.makedirs(analysis_dir, exist_ok=True)
39
40         # Directorios de salida de emapper y gráficos
41         emapper_output_dir = os.path.join(analysis_dir, '
42 emapper_output')
43         plots_dir = os.path.join(analysis_dir, 'plots')
44         os.makedirs(emapper_output_dir, exist_ok=True)
45         os.makedirs(plots_dir, exist_ok=True)
46
47         # Subdirectorios de gráficos
48         annotations_plots_dir = os.path.join(plots_dir, '
49 annotations')
50         seed_orthologs_plots_dir = os.path.join(plots_dir, '
51 seed_orthologs')
52         os.makedirs(annotations_plots_dir, exist_ok=True)
53         os.makedirs(seed_orthologs_plots_dir, exist_ok=True)
54
55         # Nombre de archivo de entrada único
```

```
50     input_filename = f"{uuid.uuid4()}.faa"
51     input_filepath = os.path.join(INPUT_DIR,
input_filename)
52     file.save(input_filepath)
53
54     # Prefijo de salida único
55     output_prefix = f"output_{uuid.uuid4()}"
56     output_prefix_path = os.path.join(emapper_output_dir
, output_prefix)
57
58     # Inicializar progreso
59     progress_data[job_id] = {"status": "Iniciado", "
progress": 0}
60
61     # Ejecutar el procesamiento en un hilo separado
62     thread = threading.Thread(target=process_emapper,
args=(
63         job_id, input_filepath, output_prefix_path,
output_prefix, options,
64         emapper_output_dir, annotations_plots_dir,
seed_orthologs_plots_dir
65     ))
66     thread.start()
67
68     return jsonify({"message": "Análisis iniciado", "
job_id": job_id}), 202
69
70     def process_emapper(job_id, input_filepath,
output_prefix_path, output_prefix, options,
71         emapper_output_dir,
annotations_plots_dir, seed_orthologs_plots_dir):
72         try:
73             # Actualizar progreso
74             progress_data[job_id]["status"] = "Ejecutando
emapper.py"
75             progress_data[job_id]["progress"] = 10
76
```

```
77         # Comando para ejecutar eggNOG-mapper
78         command = f"python3 {EMAPPER_SCRIPT} -i {
input_filepath} -o {output_prefix_path} --output_dir {
emapper_output_dir} {options}"
79
80         # Ejecutar eggNOG-mapper
81         subprocess.check_output(command, shell=True,
stderr=subprocess.STDOUT)
82
83         # Actualizar progreso
84         progress_data[job_id]["status"] = "Procesando
anotaciones"
85         progress_data[job_id]["progress"] = 50
86
87         # Archivos de salida
88         annotations_file = f"{output_prefix_path}.
emapper.annotations"
89         seed_orthologs_file = f"{output_prefix_path}.
emapper.seed_orthologs"
90
91         # Procesar anotaciones
92         progress_data[job_id]["status"] = "Generando grá
ficos de anotaciones"
93         progress_data[job_id]["progress"] = 70
94         cmd_annotations = f"python3 /app/
process_annotations.py -i {annotations_file} -o {
annotations_plots_dir}"
95         subprocess.check_output(cmd_annotations, shell=
True, stderr=subprocess.STDOUT)
96
97         # Procesar ortólogos semilla
98         progress_data[job_id]["status"] = "Generando grá
ficos de ortólogos semilla"
99         progress_data[job_id]["progress"] = 85
100        cmd_seed_orthologs = f"python3 /app/
process_seed_orthologs.py -i {seed_orthologs_file} -o {
seed_orthologs_plots_dir}"
```

```
101         subprocess.check_output(cmd_seed_orthologs ,
102                                   shell=True, stderr=subprocess.STDOUT)
103
104         # Finalizar
105         progress_data[job_id]["status"] = "Completado"
106         progress_data[job_id]["progress"] = 100
107
108     except subprocess.CalledProcessError as e:
109         # Manejo de errores
110         progress_data[job_id]["status"] = "Error"
111         progress_data[job_id]["progress"] = 100
112         progress_data[job_id]["error"] = e.output.decode
113         ("utf-8")
114     finally:
115         # Eliminar el archivo de entrada
116         os.remove(input_filepath)
117
118 @app.route('/progress/<job_id>', methods=['GET'])
119 def get_progress(job_id):
120     if job_id in progress_data:
121         return jsonify(progress_data[job_id])
122     else:
123         return jsonify({"error": "Job ID no encontrado"
124 }), 404
125
126 if __name__ == '__main__':
127     app.run(host='0.0.0.0', port=5000)
```

Listing E.1: api.py

E.2. Archivo process_annotations.py

```
1  #!/usr/bin/env python3
2  # process_annotations.py
3
```

```
4     import pandas as pd
5     import matplotlib.pyplot as plt
6     import seaborn as sns
7     import argparse
8     from collections import Counter
9     import os
10
11     parser = argparse.ArgumentParser(description='Procesar
12 anotaciones de emapper y generar gráficos.')
13     parser.add_argument('-i', '--input', required=True, help
14 = 'Archivo de anotaciones de emapper')
15     parser.add_argument('-o', '--output_dir', required=True,
16 help='Directorio de salida para los gráficos')
17     args = parser.parse_args()
18
19     if not os.path.isfile(args.input):
20         print(f"El archivo de entrada {args.input} no existe
21 .")
22         exit(1)
23
24     os.makedirs(args.output_dir, exist_ok=True)
25     sns.set(style='whitegrid', font_scale=1.2)
26
27     # Determinar la línea de encabezado en el archivo de
28 anotaciones
29     header_line = None
30     skip_rows = 0
31     with open(args.input, 'r') as f:
32         for line in f:
33             if line.startswith('##'):
34                 skip_rows += 1
35                 continue
36             elif line.startswith('#'):
37                 header_line = line[1:].strip().split('\t')
38                 skip_rows += 1
39                 break
40             else:
```

```
36         skip_rows += 1
37
38     if header_line is None:
39         print("No se encontró la línea de encabezado en el
40 archivo.")
41         exit(1)
42
43     df = pd.read_csv(args.input, sep='\t', header=None,
44 names=header_line, skiprows=skip_rows, low_memory=False)
45
46     # Gráfico de Categorías COG
47     df_cog = df.dropna(subset=['COG_category'])
48     cog_categories = ''.join(df_cog['COG_category'].tolist()
49 )
50     cog_list = list(cog_categories)
51     cog_counts = Counter(cog_list)
52     cog_df = pd.DataFrame.from_dict(cog_counts, orient='
53 index').reset_index()
54     cog_df.columns = ['COG_category', 'Count']
55
56     cog_dict = {
57         'A': 'RNA processing & modification',
58         'B': 'Chromatin structure & dynamics',
59         'C': 'Energy production & conversion',
60         'D': 'Cell cycle control, mitosis & meiosis',
61         'E': 'Amino acid transport & metabolism',
62         'F': 'Nucleotide transport & metabolism',
63         'G': 'Carbohydrate transport & metabolism',
64         'H': 'Coenzyme transport & metabolism',
65         'I': 'Lipid transport & metabolism',
66         'J': 'Translation, ribosomal structure & biogenesis'
67     },
68     {
69         'K': 'Transcription',
70         'L': 'Replication, recombination & repair',
71         'M': 'Cell wall/membrane/envelope biogenesis',
72         'N': 'Cell motility',
73     }
```

```

67         'O': 'Posttranslational modification, protein
turnover, chaperones',
68         'P': 'Inorganic ion transport & metabolism',
69         'Q': 'Secondary metabolites biosynthesis, transport
& catabolism',
70         'R': 'General function prediction only',
71         'S': 'Function unknown',
72         'T': 'Signal transduction mechanisms',
73         'U': 'Intracellular trafficking, secretion, and
vesicular transport',
74         'V': 'Defense mechanisms',
75         'W': 'Extracellular structures',
76         'Y': 'Nuclear structure',
77         'Z': 'Cytoskeleton',
78     }
79     cog_df['Description'] = cog_df['COG_category'].map(
cog_dict)
80     cog_df = cog_df.sort_values('Count', ascending=False)
81     plt.figure(figsize=(12,6))
82     sns.barplot(data=cog_df, x='COG_category', y='Count',
palette='viridis')
83     plt.title('Distribución de Categorías COG')
84     plt.xlabel('Categoría COG')
85     plt.ylabel('Número de Proteínas')
86     plt.tight_layout()
87     plt.savefig(os.path.join(args.output_dir, '
cog_categories.png'), dpi=300)
88     plt.close()
89
90     # Términos GO Más Comunes
91     df_go = df.dropna(subset=['GOs'])
92     go_terms = df_go['GOs'].str.split(',', expand=True).
stack().reset_index(level=1, drop=True)
93     go_terms.name = 'GO_term'
94     go_counts = go_terms.value_counts().reset_index()
95     go_counts.columns = ['GO_term', 'Count']
96     top_go = go_counts.head(20)

```

```
97     plt.figure(figsize=(10,8))
98     sns.barplot(data=top_go, y='GO_term', x='Count', palette
='magma')
99     plt.title('Top 20 Términos GO Más Comunes')
100    plt.xlabel('Número de Proteínas')
101    plt.ylabel('Término GO')
102    plt.tight_layout()
103    plt.savefig(os.path.join(args.output_dir, 'top_go_terms.
png'), dpi=300)
104    plt.close()
105
106    # Vías KEGG Más Comunes
107    df_kegg = df.dropna(subset=['KEGG_Pathway'])
108    kegg_pathways = df_kegg['KEGG_Pathway'].str.split(',',
expand=True).stack().reset_index(level=1, drop=True)
109    kegg_pathways.name = 'KEGG_Pathway'
110    kegg_counts = kegg_pathways.value_counts().reset_index()
111    kegg_counts.columns = ['KEGG_Pathway', 'Count']
112    top_kegg = kegg_counts.head(10)
113    plt.figure(figsize=(10,6))
114    sns.barplot(data=top_kegg, y='KEGG_Pathway', x='Count',
palette='cividis')
115    plt.title('Top 10 Vías KEGG Más Comunes')
116    plt.xlabel('Número de Proteínas')
117    plt.ylabel('Vía KEGG')
118    plt.tight_layout()
119    plt.savefig(os.path.join(args.output_dir, '
top_kegg_pathways.png'), dpi=300)
120    plt.close()
121
122    # Familias PFAM Más Comunes
123    df_pfam = df.dropna(subset=['PFAMs'])
124    pfam_families = df_pfam['PFAMs'].str.split(',', expand=
True).stack().reset_index(level=1, drop=True)
125    pfam_families.name = 'PFAM'
126    pfam_counts = pfam_families.value_counts().reset_index()
127    pfam_counts.columns = ['PFAM', 'Count']
```

```

128     top_pfam = pfam_counts.head(15)
129     plt.figure(figsize=(10,8))
130     sns.barplot(data=top_pfam, y='PFAM', x='Count', palette=
'plasma')
131     plt.title('Top 15 Familias PFAM Más Comunes')
132     plt.xlabel('Número de Proteínas')
133     plt.ylabel('Familia PFAM')
134     plt.tight_layout()
135     plt.savefig(os.path.join(args.output_dir, '
top_pfam_families.png'), dpi=300)
136     plt.close()
137
138     # Números EC Más Comunes
139     df_ec = df.dropna(subset=['EC'])
140     ec_numbers = df_ec['EC'].str.split(',', expand=True).
stack().reset_index(level=1, drop=True)
141     ec_numbers.name = 'EC_number'
142     ec_counts = ec_numbers.value_counts().reset_index()
143     ec_counts.columns = ['EC_number', 'Count']
144     top_ec = ec_counts.head(10)
145     plt.figure(figsize=(10,6))
146     sns.barplot(data=top_ec, y='EC_number', x='Count',
palette='inferno')
147     plt.title('Top 10 Números EC Más Comunes')
148     plt.xlabel('Número de Proteínas')
149     plt.ylabel('Número EC')
150     plt.tight_layout()
151     plt.savefig(os.path.join(args.output_dir, '
top_ec_numbers.png'), dpi=300)
152     plt.close()
153

```

Listing E.2: process_annotations.py

E.3. Archivo process_seed_orthologs.py

```
1  #!/usr/bin/env python3
2  # process_seed_orthologs.py
3
4  import pandas as pd
5  import matplotlib.pyplot as plt
6  import seaborn as sns
7  import argparse
8  import os
9
10 parser = argparse.ArgumentParser(description='Procesar
11 ortólogos semilla de emapper y generar gráficos.')
12 parser.add_argument('-i', '--input', required=True, help
13 = 'Archivo de ortólogos semilla de emapper')
14 parser.add_argument('-o', '--output_dir', required=True,
15 help='Directorio de salida para los gráficos')
16 args = parser.parse_args()
17
18 if not os.path.isfile(args.input):
19     print(f"El archivo de entrada {args.input} no existe
20 .")
21     exit(1)
22
23 os.makedirs(args.output_dir, exist_ok=True)
24 sns.set(style='whitegrid', font_scale=1.2)
25
26 # Determinar la línea de encabezado en el archivo de ort
27 ólogos semilla
28 header_line = None
29 skip_rows = 0
30 with open(args.input, 'r') as f:
31     for line in f:
32         if line.startswith('##'):
33             skip_rows += 1
34             continue
35         elif line.startswith('#'):
36             header_line = line[1:].strip().split('\t')
```

```
32         skip_rows += 1
33         break
34     else:
35         skip_rows += 1
36
37     if header_line is None:
38         print("No se encontró la línea de encabezado en el
39         archivo.")
40         exit(1)
41
42     df = pd.read_csv(args.input, sep='\t', header=None,
43     names=header_line, skiprows=skip_rows)
44
45     numerical_cols = ['evalue', 'bitscore', 'qstart', 'qend',
46     , 'sstart', 'send', 'pident', 'qcov', 'scov']
47     df[numerical_cols] = df[numerical_cols].apply(pd.
48     to_numeric, errors='coerce')
49     df = df.dropna(subset=numerical_cols)
50
51     # Distribución de e-values
52     plt.figure(figsize=(10,6))
53     sns.histplot(df['evalue'], bins=50, log_scale=(True,
54     False))
55     plt.title('Distribución de e-values')
56     plt.xlabel('e-value')
57     plt.ylabel('Frecuencia')
58     plt.tight_layout()
59     plt.savefig(os.path.join(args.output_dir, '
60     evalue_distribution.png'), dpi=300)
61     plt.close()
62
63     # Bitscore vs. Porcentaje de Identidad
64     plt.figure(figsize=(10,6))
65     sns.scatterplot(data=df, x='bitscore', y='pident')
66     plt.title('Bitscore vs. Porcentaje de Identidad')
67     plt.xlabel('Bitscore')
68     plt.ylabel('Porcentaje de Identidad (%)')
```

```
63     plt.tight_layout()
64     plt.savefig(os.path.join(args.output_dir, '
bitscore_vs_pident.png'), dpi=300)
65     plt.close()
66
67     # Bitscore por Secuencia de Consulta
68     plt.figure(figsize=(12,8))
69     df_sorted = df.sort_values(by='bitscore', ascending=
False)
70     sns.barplot(data=df_sorted, x='qseqid', y='bitscore')
71     plt.title('Bitscore por Secuencia de Consulta')
72     plt.xlabel('ID de Secuencia de Consulta')
73     plt.ylabel('Bitscore')
74     plt.xticks(rotation=90)
75     plt.tight_layout()
76     plt.savefig(os.path.join(args.output_dir, '
bitscore_per_sequence.png'), dpi=300)
77     plt.close()
78
79     # Mapa de Calor de Coberturas
80     coverage_data = df[['qseqid', 'qcov', 'scov']]
81     coverage_data = coverage_data.melt(id_vars='qseqid',
var_name='Tipo', value_name='Cobertura')
82     coverage_data = coverage_data.drop_duplicates(subset=['
qseqid', 'Tipo'])
83     pivot_table = coverage_data.pivot(index='qseqid',
columns='Tipo', values='Cobertura')
84
85     plt.figure(figsize=(10,12))
86     sns.heatmap(pivot_table, annot=False, cmap='viridis')
87     plt.title('Mapa de Calor de Coberturas')
88     plt.xlabel('Tipo de Cobertura')
89     plt.ylabel('ID de Secuencia de Consulta')
90     plt.tight_layout()
91     plt.savefig(os.path.join(args.output_dir, '
coverage_heatmap.png'), dpi=300)
92     plt.close()
```

```
93
94     # Matriz de Correlación
95     plt.figure(figsize=(8,6))
96     corr_matrix = df[['evalue', 'bitscore', 'pident', 'qcov',
97 , 'scov']].corr()
98     sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
99     plt.title('Matriz de Correlación')
100    plt.tight_layout()
101    plt.savefig(os.path.join(args.output_dir, '
102    correlation_matrix.png'), dpi=300)
103    plt.close()
```

Listing E.3: process_seed_orthologs.py

E.4. Archivo Dockerfile

```
1     # Imagen base de Python
2     FROM python:3.8-slim
3
4     # Instalar dependencias del sistema
5     RUN apt-get update && apt-get install -y \
6         build-essential \
7         wget \
8         libfreetype6-dev \
9         libpng-dev \
10        libopenblas-dev \
11        && rm -rf /var/lib/apt/lists/*
12
13    # Copiar eggno-mapper
14    COPY eggno-mapper-2.1.12 /app/eggno-mapper-2.1.12
15
16    # Fijar versión de numpy
17    RUN pip install numpy==1.21.2
18
19    # Instalar EggNOG-mapper
```

```
20 WORKDIR /app/eggnog-mapper-2.1.12
21 RUN pip install -r requirements.txt && \
22     python3 setup.py install && \
23     python3 download_eggnog_data.py -y
24
25 # Instalar Flask y otras dependencias
26 RUN pip install Flask pandas matplotlib seaborn
27 wordcloud
28
29 # Copiar scripts de la API y procesamiento
30 COPY api.py /app/api.py
31 COPY process_annotations.py /app/process_annotations.py
32 COPY process_seed_orthologs.py /app/
33 process_seed_orthologs.py
34
35 # Exponer el puerto de la API
36 EXPOSE 5000
37
38 # Comando para iniciar la API
39 CMD ["python3", "/app/api.py"]
```

Listing E.4: Dockerfile

Este código fuente proporciona una visión integral de la lógica interna del backend, la integración con las herramientas bioinformáticas y la generación de resultados visuales. Además, demuestra el uso de buenas prácticas en la organización de módulos, validación de entradas, manejo de errores y uso de contenedores para asegurar la reproducibilidad y escalabilidad del sistema.