

# Proyecto Bimestral

Integrantes

Ana Guamán, Steven Neira, Ruben Jimenez

# Búsqueda de un tema



dinosaurs robotics IA



Motivación de elección

Escogimos este tema por la fascinación de un integrante del grupo por la Paleontología, nos pareció sumamente interesante el hecho de poder encontrar como se usa las tecnologías emergentes en identificar huellas de fosiles, escaneos, reconstrucción y representación en 3D para interactuar con dichos fósiles.



# Retos y dificultades

El primer reto que nos encontramos fue la cantidad de resultados que obtuvimos al buscar en Semantic Scholar, por lo que tuvimos que usar filtros y posteriormente cambiar el texto de la búsqueda.

Ahora con la nueva búsqueda encontramos 1000 resultados pero enfocados en plenamente en el desarrollo de la IA y robots, aun que nuestro objetivo [los dinosaurios] sigue en menor medida como área beneficiada por lo anterior mencionado



# API DE SEMANTIC SCHOLAR

## Request a Semantic Scholar API Key

We prioritize requests for academic and research institutions, nonprofit organizations, and government entities. We review all other requests on a case-by-case basis in accordance with applicable laws and regulations. Thank you for your patience!

If you need to regain access to your previous inactive key, please submit a new request.

First name\*

Last name\*

Email\*

Please use an academic or corporate email. If your key is intended to be used for a company/organization, consider using a persistent identity so that you can receive important notifications.

Affiliation\*

Full name of company or other formal affiliation entity. Do not abbreviate.

Affiliation website URL\*

Link to company or other formal affiliation website.

Country/Region\*

Please Select

Country/Region

If you are based in Ukraine, please specify your region.

Antes de empezar con la extracción necesitábamos la api, para lo cual cada uno de los integrantes llenamos el formulario y al poco tiempo nos llegó la api key. Para la extracción nos dividimos para sacar datos de 300 - 350 - 350

```
GRUPO_INICIO = 351      # <- CAMBIAR AQUI
GRUPO_FIN = 700        # <- CAMBIAR AQUI

# =====

import requests
import time
import pandas as pd
from datetime import datetime

# API Key
api_key = "TrYlRjBSapuc29CCTztnQiih80BzF2j8narAo60"
headers = {"x-api-key": api_key}
```

# CONTROL DE ERRORES

```
def safe_get(url, params=None, max_retries=3):
    for attempt in range(max_retries):
        try:
            log_request()
            response = requests.get(url, headers=headers, params=params, timeout=20)

            if response.status_code == 200:
                return response.json()
            elif response.status_code == 429:
                wait_time = 65
                print(f"Rate limit hit (intento {attempt+1}/{max_retries}), esperando {wait_time}s...")
                time.sleep(wait_time)
```

Tenemos dos funciones de extracción que nos ayudan a sacar la información completa del autor y del paper.

Dentro de nuestro código de extracción tenemos control de errores, para que no se corte el flujo, ya que obtenemos los csv al final

## EXTRACION DESDE LA API

```
def get_author_details(author_id):
    url = f"https://api.semanticscholar.org/graph/v1/author/{author_id}"
    params = {"fields": "name,affiliations,homepage,paperCount,citationCount,hIndex"}
    return safe_get(url, params)

def get_paper_details(paper_id):
    url = f"https://api.semanticscholar.org/graph/v1/paper/{paper_id}"
    params = {
        "fields": "title,authors,abstract,year,publicationDate,externalIds,citationCount,references,citations",
    }

    data = safe_get(url, params)

    # Validar y enriquecer fieldsOfStudy si esta vacio
    if data and 'paperId' in data:
        fields = data.get('fieldsOfStudy') or data.get('s2FieldsOfStudy')
        if not fields or len(fields) == 0:
            pub_venue = data.get('publicationVenue', {})
            if pub_venue and isinstance(pub_venue, dict):
                data['fieldsOfStudy'] = [pub_venue.get('name', 'N/A')] if pub_venue.get('name') else []

return data
```

# Funciones de almacenamiento y enriquecimiento

Ahora nuestras funciones mas importantes:

01

process\_paper()

Aquí es donde procesamos cada paper: limpiamos su información, registramos autores y organizaciones, y guardamos sus citaciones y referencias para construir toda la red.

02

add\_enriched\_citation()

Guarda citaciones añadiendo títulos, años y ids

03

enrich\_authors\_data()

Se pide información de cada autor, se agregan observaciones y metricas

04

get\_seed\_papers\_for\_group()

Aquí es donde hacemos la búsqueda para obtener los resultados que nos dividimos 300-350-350

# Limpieza y generación de **RDF**

definimos los archivos CSV de entrada y los archivos de salida

```
12 # Archivos de entrada
13 INPUT_PAPERS = 'papers_0_350.csv'
14 INPUT_AUTHORS = 'authors_0_350.csv'
15 INPUT_PAPER_AUTHORS = 'paper_authors_0_350.csv'
16 INPUT_CITATIONS = 'citations_0_350.csv'
17 INPUT_OBSERVATIONS = 'observations_0_350.csv'
18
19 # Archivos de salida CSV (limpios)
20 OUTPUT_PAPERS_CSV = 'papers_CLEAN.csv'
21 OUTPUT_AUTHORS_CSV = 'authors_CLEAN.csv'
22 OUTPUT_PAPER_AUTHORS_CSV = 'paper_authors_CLEAN.csv'
23 OUTPUT_CITATIONS_CSV = 'citations_CLEAN.csv'
24 OUTPUT_OBSERVATIONS_CSV = 'observations_CLEAN.csv'
25
26 # Archivos de salida TTL (separados por dominio)
27 OUTPUT_TTL_PAPERS = 'papers.ttl'
28 OUTPUT_TTL_AUTHORS = 'authors.ttl'
29 OUTPUT_TTL_CITATIONS = 'citations.ttl'
30 OUTPUT_TTL_VENUES = 'venues.ttl'
31 OUTPUT_TTL_OBSERVATIONS = 'observations.ttl'
32 OUTPUT_TTL_FIELDS = 'fields_of_study.ttl'
33
34 OUTPUT_REPORT = 'limpieza_reporte.txt'
35
```

prefijos utilizados para crear URIs en los archivos RDF

```
# Namespace base para los datos
DATA = Namespace("http://example.org/data/")
VOCAB = Namespace("http://example.org/vocab/")

# Namespaces estandar
SCHEMA = Namespace("http://schema.org/")
DBR = Namespace("http://dbpedia.org/resource/")
```

# Estrategias Clave

01

## LIMPIEZA

En esta etapa se aplicaron técnicas de limpieza de datos para asegurar la calidad, consistencia y validez de la información antes de transformarla a RDF.

Se eliminaron filas sin identificadores, se removieron duplicados, se normalizaron textos, se corrigieron fechas, se validaron formatos, y se filtraron relaciones inválidas entre entidades.

02

## CONVERSIÓN A RDF

Se realizó la transformación completa del dataset a un grafo RDF utilizando ontologías estándar, incluyendo Schema.org, DCTERMS, SKOS y XSD.

Cada registro fue representado como un recurso identificado con un URI único, y se generaron triples RDF que describen entidades, atributos y relaciones entre ellas.

03

## Generación de grafos consistentes

Se construyeron objetos Graph de RDFLib para cada tipo de entidad y se añadieron triples validados. Finalmente, cada grafo fue serializado en formato TTL para exportarlo a GraphDB.

# GRAFO



Al final obtenemos el grafo probado en graphDB, lo realizamos haciendo las consultas en sparkQL

