Product Requirements Document (PRD)

# PRD — Multi-Agent Research Portfolio Dashboard (three-agent framework)

## 1. Overview / Vision
Build a modular multi-agent system that automatically fetches publication data from a researcher's Google Scholar profile, enriches and normalizes that data, and curates an interactive, filterable research-portfolio dashboard. The system uses:

- Supervisor agent: orchestrates workflow, validation, retries.
- Fetcher agent: given a Google Scholar profile link + TAVILY_API_KEY, fetches publications, citations, coauthors, and PDFs (where available).
- Dashboard agent: prepares cleaned data, analytics, visualizations, and front-end artifacts (React/Node) and drives incremental updates to the UI.

The app: backend in Python (FastAPI), frontend in Node.js (Next.js or React + API client), deployed on Render. Use Claude/agent runner for agent orchestration and Tavily to retrieve web data.

## 2. Goals & Success Criteria
- Fully automated pipeline from link → live dashboard.
- Accurate extraction of: title, authors, venue, year, abstract (if available), citation count, coauthors, DOI, PDF link, and scraped metadata (keywords, references).
- Dashboard must be interactive (filters: year, topic, coauthor; charts: timeline, citation trend, top venues).
- Secure handling of keys (no hardcoding).
- CI/CD with Render deployment: backend & frontend auto-deploy on push to main.

Success metrics:
- Data completeness: ≥90% of publications have title + year + at least one author.
- Latency: fetch + process each profile within practical SLA.
- Robustness: rescraping handles minor DOM changes and missing fields.

## 3. Personas
- Primary: Researcher (you).
- Secondary: Hiring committee / collaborators.
- Developer (you / teammates).

## 4. Core Features (MVP)
- Provide Google Scholar profile link and TAVILY_API_KEY.
- Automated fetch & parse of publications + metadata.

- Normalization & deduplication.
- Store data in PostgreSQL + optional vector store for semantic search.
- Dashboard UI: list, search, filters, charts, downloadable CV (PDF).
- Admin panel: trigger manual re-scrape, view pipeline logs.

## 5. Non-functional requirements
- Secure storage of API keys.
- Modular agents with well-defined contracts.
- Observability: logs, metrics.
- Extensible: add more sources later.

## 6. Data model (core fields)
```
{
  "id": "string",
  "title": "string",
  "authors": ["First Last"],
  "year": 2024,
  "venue": "Conference / Journal",
  "abstract": "string|null",
  "doi": "string|null",
  "pdf_url": "string|null",
  "citations": 123,
  "scholar_link": "https://scholar.google.com/...",
  "scrape_timestamp": "2025-10-04T11:00:00Z",
  "keywords": ["nlp","security"],
  "coauthors": ["A Name"],
  "raw_html_snippet": "string"
}
```

## 7. Architecture & Components
- Agents: Supervisor, Fetcher, Dashboard.
- Backend: FastAPI, PostgreSQL, optional Redis, Celery/RQ.
- Frontend: Next.js/React.
- Deployment: Render.

## 8. Security & Compliance
- Never log API keys.
- Use HTTPS and env variables.

## 9. Observability & Testing
- Structured logs.
- Unit, integration, and E2E tests.

## 10. Deliverables / Milestones
- A: Agent orchestration + Fetcher prototype.
- B: Backend API and persistence.
- C: Dashboard agent + frontend skeleton.
- D: Full CI/CD & Render deployment.