Table of Contents

# F. STREAMLIT APP

## F.1 Introducción

Este anexo documenta la aplicación web interactiva desarrollada con Streamlit para la exploración y visualización de los resultados experimentales del sistema RAG. La aplicación proporciona una interfaz intuitiva para analizar el rendimiento de los diferentes modelos de embedding, visualizar métricas comparativas, y explorar casos específicos de recuperación de documentos.

## F.2 Arquitectura de la Aplicación

### F.2.1 Estructura del Proyecto Streamlit

src/apps/ # Aplicaciones Streamlit del proyecto  
├── main\_qa\_app.py # Aplicación principal de Q&A  
├── cumulative\_metrics\_results\_matplotlib.py # Visualización de resultados  
├── comparison\_page.py # Comparación de modelos  
├── cumulative\_comparison.py # Comparación acumulativa  
├── batch\_queries\_page.py # Procesamiento de consultas en lote  
├── data\_analysis\_page.py # Análisis de datos experimentales  
└── question\_answer\_comparison.py # Comparación de respuestas  
  
src/ui/ # Interfaces de usuario compartidas  
├── display.py # Funciones de visualización  
├── enhanced\_metrics\_display.py # Visualización de métricas avanzadas  
├── metrics\_display.py # Visualización básica de métricas  
└── pdf\_generator.py # Generación de reportes PDF  
  
.streamlit/ # Configuración de Streamlit (directorio raíz)  
└── config.toml # Configuración global de Streamlit

## F.3 Funcionalidades Principales

### F.3.1 Página Principal (Dashboard)

#### F.3.1.1 Resumen Ejecutivo

La página principal presenta un dashboard con las métricas clave del sistema:

# Métricas principales mostradas  
col1, col2, col3, col4 = st.columns(4)  
  
with col1:  
 st.metric(  
 label="📊 Modelos Evaluados",   
 value="4",  
 help="Ada, MPNet, MiniLM, E5-Large"  
 )  
  
with col2:  
 st.metric(  
 label="📋 Preguntas Evaluadas",   
 value="11",  
 help="Por modelo, total 44 evaluaciones"  
 )  
  
with col3:  
 st.metric(  
 label="📚 Documentos Indexados",   
 value="187,031",  
 help="Chunks de documentación Azure"  
 )  
  
with col4:  
 st.metric(  
 label="⏱️ Tiempo Total",   
 value="12.9 min",  
 help="774.78 segundos de evaluación"  
 )

#### F.3.1.2 Selector de Archivos de Resultados

La aplicación permite cargar diferentes archivos de resultados experimentales:

# Selector de archivos de resultados  
results\_files = [  
 "cumulative\_results\_1753578255.json",  
 "cumulative\_results\_20250731\_140825.json"  
]  
  
selected\_file = st.selectbox(  
 "📁 Seleccionar archivo de resultados:",  
 results\_files,  
 help="Selecciona el archivo de resultados experimentales a analizar"  
)

### F.3.2 Comparación de Modelos

#### F.3.2.1 Tabla Comparativa Interactiva

def create\_comparison\_table():  
 """Crea tabla comparativa de modelos con métricas clave"""  
   
 comparison\_data = {  
 'Modelo': ['Ada', 'MPNet', 'MiniLM', 'E5-Large'],  
 'Dimensiones': [1536, 768, 384, 1024],  
 'Precision@5': [0.055, 0.055, 0.036, 0.000],  
 'Recall@5': [0.273, 0.273, 0.182, 0.000],  
 'NDCG@5': [0.162, 0.189, 0.103, 0.000],  
 'BERTScore F1': [0.732, 0.739, 0.729, 0.739],  
 'Faithfulness': [0.482, 0.518, 0.509, 0.591]  
 }  
   
 df = pd.DataFrame(comparison\_data)  
   
 # Aplicar formato condicional  
 st.dataframe(  
 df.style.background\_gradient(subset=['Precision@5', 'Recall@5']),  
 use\_container\_width=True  
 )

#### F.3.2.2 Gráfico Radar Comparativo

def create\_radar\_chart():  
 """Crea gráfico radar para comparación multi-dimensional"""  
   
 fig = go.Figure()  
   
 metrics = ['Precision@5', 'Recall@5', 'NDCG@5', 'BERTScore F1', 'Faithfulness']  
   
 for model\_name, values in model\_data.items():  
 fig.add\_trace(go.Scatterpolar(  
 r=values,  
 theta=metrics,  
 fill='toself',  
 name=model\_name,  
 line=dict(width=2)  
 ))  
   
 fig.update\_layout(  
 polar=dict(  
 radialaxis=dict(  
 visible=True,  
 range=[0, 1]  
 )  
 ),  
 showlegend=True,  
 title="📊 Comparación Multi-Dimensional de Modelos"  
 )  
   
 st.plotly\_chart(fig, use\_container\_width=True)

### F.3.3 Análisis de Métricas

#### F.3.3.1 Visualización de Impacto del Reranking

def plot\_reranking\_impact():  
 """Visualiza el impacto del reranking por modelo"""  
   
 models = ['Ada', 'MPNet', 'MiniLM', 'E5-Large']  
 pre\_reranking = [0.126, 0.108, 0.091, 0.000]  
 post\_reranking = [0.162, 0.189, 0.103, 0.000]  
   
 x = np.arange(len(models))  
 width = 0.35  
   
 fig, ax = plt.subplots(figsize=(10, 6))  
   
 bars1 = ax.bar(x - width/2, pre\_reranking, width,   
 label='Pre-Reranking', alpha=0.8)  
 bars2 = ax.bar(x + width/2, post\_reranking, width,  
 label='Post-Reranking', alpha=0.8)  
   
 # Añadir etiquetas de mejora porcentual  
 for i, (pre, post) in enumerate(zip(pre\_reranking, post\_reranking)):  
 if pre > 0:  
 improvement = ((post - pre) / pre) \* 100  
 ax.text(i, post + 0.01, f'+{improvement:.1f}%',   
 ha='center', va='bottom', fontweight='bold')  
   
 ax.set\_xlabel('Modelos')  
 ax.set\_ylabel('NDCG@5')  
 ax.set\_title('🎯 Impacto del CrossEncoder Reranking')  
 ax.set\_xticks(x)  
 ax.set\_xticklabels(models)  
 ax.legend()  
   
 st.pyplot(fig)

#### F.3.3.2 Análisis Estadístico (Wilcoxon)

def display\_statistical\_analysis():  
 """Muestra resultados de tests estadísticos"""  
   
 st.subheader("📊 Análisis de Significancia Estadística")  
   
 # Cargar resultados de Wilcoxon  
 wilcoxon\_data = pd.read\_csv('wilcoxon\_test\_results.csv')  
   
 # Filtrar por métrica seleccionada  
 metric = st.selectbox(  
 "Seleccionar métrica:",  
 ['precision@5', 'recall@5', 'f1@5', 'ndcg@5']  
 )  
   
 filtered\_data = wilcoxon\_data[wilcoxon\_data['metric'] == metric]  
   
 # Crear heatmap de p-valores  
 pivot\_table = filtered\_data.pivot\_table(  
 values='p\_value',   
 index='model1',   
 columns='model2'  
 )  
   
 fig, ax = plt.subplots(figsize=(8, 6))  
 sns.heatmap(pivot\_table, annot=True, cmap='RdYlBu\_r',   
 center=0.05, ax=ax)  
 ax.set\_title(f'P-valores Test de Wilcoxon - {metric.upper()}')  
   
 st.pyplot(fig)  
   
 # Interpretación  
 significant\_pairs = filtered\_data[filtered\_data['significant'] == True]  
   
 if len(significant\_pairs) == 0:  
 st.warning("⚠️ No se encontraron diferencias estadísticamente significativas (p > 0.05)")  
 else:  
 st.success(f"✅ {len(significant\_pairs)} comparaciones estadísticamente significativas")

### F.3.4 Explorador de Consultas

#### F.3.4.1 Búsqueda Interactiva

def create\_query\_explorer():  
 """Interfaz para explorar consultas específicas"""  
   
 st.subheader("🔍 Explorador de Consultas")  
   
 # Selector de consulta  
 query\_options = load\_query\_list()  
 selected\_query = st.selectbox(  
 "Seleccionar consulta:",  
 query\_options,  
 help="Elige una consulta para ver resultados detallados"  
 )  
   
 # Selector de modelo  
 model\_options = ['Ada', 'MPNet', 'MiniLM', 'E5-Large']  
 selected\_model = st.selectbox(  
 "Seleccionar modelo:",  
 model\_options  
 )  
   
 # Mostrar resultados  
 if st.button("🔍 Buscar"):  
 results = get\_query\_results(selected\_query, selected\_model)  
 display\_query\_results(results)  
  
def display\_query\_results(results):  
 """Muestra resultados detallados de una consulta"""  
   
 st.write(f"\*\*Consulta:\*\* {results['query']}")  
 st.write(f"\*\*Modelo:\*\* {results['model']}")  
   
 # Métricas de la consulta  
 col1, col2, col3 = st.columns(3)  
   
 with col1:  
 st.metric("Precision@5", f"{results['precision\_5']:.3f}")  
 with col2:  
 st.metric("NDCG@5", f"{results['ndcg\_5']:.3f}")  
 with col3:  
 st.metric("MRR", f"{results['mrr']:.3f}")  
   
 # Top 10 documentos recuperados  
 st.subheader("📋 Top 10 Documentos Recuperados")  
   
 for i, doc in enumerate(results['top\_documents'][:10], 1):  
 with st.expander(f"#{i} - Score: {doc['score']:.3f}"):  
 st.write(f"\*\*Título:\*\* {doc['title']}")  
 st.write(f"\*\*URL:\*\* {doc['url']}")  
 st.write(f"\*\*Snippet:\*\* {doc['content'][:200]}...")  
   
 # Indicador de relevancia  
 if doc['is\_relevant']:  
 st.success("✅ Documento relevante según ground truth")  
 else:  
 st.info("ℹ️ Documento no marcado como relevante")

### F.3.5 Visualizaciones Avanzadas

#### F.3.5.1 Distribución de Scores de Similitud

def plot\_similarity\_distribution():  
 """Visualiza distribución de scores de similitud por modelo"""  
   
 fig, axes = plt.subplots(2, 2, figsize=(12, 8))  
 fig.suptitle('📊 Distribución de Scores de Similitud Coseno')  
   
 models = ['Ada', 'MPNet', 'MiniLM', 'E5-Large']  
   
 for i, model in enumerate(models):  
 ax = axes[i//2, i%2]  
   
 # Obtener scores del modelo  
 scores = get\_similarity\_scores(model)  
   
 if len(scores) > 0:  
 ax.hist(scores, bins=20, alpha=0.7, edgecolor='black')  
 ax.set\_title(f'{model}')  
 ax.set\_xlabel('Cosine Similarity Score')  
 ax.set\_ylabel('Frequency')  
   
 # Añadir línea vertical para el promedio  
 mean\_score = np.mean(scores)  
 ax.axvline(mean\_score, color='red', linestyle='--',   
 label=f'Media: {mean\_score:.3f}')  
 ax.legend()  
 else:  
 ax.text(0.5, 0.5, 'No hay datos\ndisponibles',   
 ha='center', va='center', transform=ax.transAxes)  
 ax.set\_title(f'{model} - Sin datos')  
   
 plt.tight\_layout()  
 st.pyplot(fig)

#### F.3.5.2 Análisis de Correlación entre Métricas

def plot\_metrics\_correlation():  
 """Visualiza correlaciones entre diferentes métricas"""  
   
 # Crear matriz de correlación  
 metrics\_data = prepare\_correlation\_data()  
 correlation\_matrix = metrics\_data.corr()  
   
 # Crear heatmap  
 fig, ax = plt.subplots(figsize=(10, 8))  
   
 mask = np.triu(np.ones\_like(correlation\_matrix, dtype=bool))  
   
 sns.heatmap(correlation\_matrix,   
 mask=mask,  
 annot=True,   
 cmap='coolwarm',   
 center=0,  
 square=True,  
 fmt='.3f',  
 ax=ax)  
   
 ax.set\_title('🔗 Matriz de Correlación entre Métricas')  
   
 st.pyplot(fig)  
   
 # Interpretación de correlaciones importantes  
 st.subheader("🔍 Interpretación de Correlaciones")  
   
 high\_correlations = find\_high\_correlations(correlation\_matrix)  
   
 for correlation in high\_correlations:  
 if correlation['value'] > 0.7:  
 st.success(f"✅ \*\*{correlation['metric1']}\*\* y \*\*{correlation['metric2']}\*\*: "  
 f"Correlación alta ({correlation['value']:.3f})")  
 elif correlation['value'] < -0.7:  
 st.warning(f"⚠️ \*\*{correlation['metric1']}\*\* y \*\*{correlation['metric2']}\*\*: "  
 f"Correlación negativa fuerte ({correlation['value']:.3f})")

### F.3.6 Exportación de Reportes

#### F.3.6.1 Generación de Reportes PDF

def generate\_pdf\_report():  
 """Genera reporte PDF con todos los resultados"""  
   
 st.subheader("📄 Generar Reporte PDF")  
   
 # Opciones de reporte  
 include\_sections = st.multiselect(  
 "Seleccionar secciones a incluir:",  
 [  
 "Resumen Ejecutivo",  
 "Comparación de Modelos",   
 "Análisis de Métricas",  
 "Casos de Ejemplo",  
 "Análisis Estadístico",  
 "Recomendaciones"  
 ],  
 default=["Resumen Ejecutivo", "Comparación de Modelos"]  
 )  
   
 if st.button("📄 Generar Reporte"):  
 with st.spinner("Generando reporte PDF..."):  
 pdf\_buffer = create\_pdf\_report(include\_sections)  
   
 st.download\_button(  
 label="📥 Descargar Reporte PDF",  
 data=pdf\_buffer,  
 file\_name=f"reporte\_sistema\_rag\_{datetime.now().strftime('%Y%m%d\_%H%M%S')}.pdf",  
 mime="application/pdf"  
 )

#### F.3.6.2 Exportación de Datos

def export\_data\_section():  
 """Sección para exportar datos experimentales"""  
   
 st.subheader("💾 Exportar Datos")  
   
 export\_format = st.radio(  
 "Formato de exportación:",  
 ["CSV", "JSON", "Excel"]  
 )  
   
 data\_type = st.selectbox(  
 "Tipo de datos:",  
 [  
 "Métricas por modelo",  
 "Resultados por consulta",  
 "Análisis estadístico",  
 "Datos completos"  
 ]  
 )  
   
 if st.button("💾 Exportar"):  
 data = prepare\_export\_data(data\_type)  
   
 if export\_format == "CSV":  
 csv\_buffer = data.to\_csv(index=False)  
 st.download\_button(  
 "📥 Descargar CSV",  
 csv\_buffer,  
 f"datos\_{data\_type.replace(' ', '\_')}\_{datetime.now().strftime('%Y%m%d')}.csv"  
 )  
 elif export\_format == "JSON":  
 json\_buffer = data.to\_json(orient='records', indent=2)  
 st.download\_button(  
 "📥 Descargar JSON",   
 json\_buffer,  
 f"datos\_{data\_type.replace(' ', '\_')}\_{datetime.now().strftime('%Y%m%d')}.json"  
 )

## F.4 Configuración y Despliegue

### F.4.1 Configuración de Streamlit

**Archivo .streamlit/config.toml:**

[global]  
dataFrameSerialization = "legacy"  
  
[server]  
port = 8501  
address = "localhost"  
maxUploadSize = 200  
enableCORS = true  
enableXsrfProtection = true  
  
[browser]  
gatherUsageStats = false  
serverAddress = "localhost"  
serverPort = 8501  
  
[theme]  
primaryColor = "#0078d4" # Azure blue  
backgroundColor = "#ffffff" # White background  
secondaryBackgroundColor = "#f5f5f5" # Light gray  
textColor = "#000000" # Black text  
font = "sans serif"  
  
[logger]  
level = "info"

### F.4.2 Variables de Ambiente

# .env para Streamlit  
STREAMLIT\_SERVER\_PORT=8501  
STREAMLIT\_SERVER\_ADDRESS=localhost  
STREAMLIT\_THEME\_PRIMARY\_COLOR=#0078d4  
  
# Paths de datos  
RESULTS\_DATA\_PATH=./data/  
CHROMADB\_PATH=/Users/haroldgomez/chromadb2  
  
# APIs (si se requieren)  
OPENAI\_API\_KEY=your\_api\_key\_here

### F.4.3 Comandos de Ejecución

# Desarrollo local - Aplicación principal Q&A  
streamlit run src/apps/main\_qa\_app.py  
  
# Aplicación de resultados experimentales  
streamlit run src/apps/cumulative\_metrics\_results\_matplotlib.py  
  
# Aplicación de comparación de modelos  
streamlit run src/apps/comparison\_page.py  
  
# Con configuración específica  
streamlit run src/apps/main\_qa\_app.py --server.port 8502  
  
# Modo debug  
streamlit run src/apps/main\_qa\_app.py --logger.level debug

## F.5 Funcionalidades de Usuario

### F.5.1 Navegación Intuitiva

* **Sidebar navigation:** Navegación entre páginas mediante sidebar
* **Breadcrumbs:** Indicadores de ubicación actual
* **Search functionality:** Búsqueda rápida de consultas y documentos

### F.5.2 Interactividad

* **Filtros dinámicos:** Filtrar resultados por modelo, métrica, rango de fechas
* **Zoom en gráficos:** Gráficos interactivos con Plotly
* **Tooltips informativos:** Ayuda contextual en métricas y visualizaciones

### F.5.3 Personalización

* **Temas:** Soporte para modo claro/oscuro
* **Exportación:** Múltiples formatos de exportación
* **Configuración:** Preferencias de usuario persistentes

## F.6 Casos de Uso de la Aplicación

### F.6.1 Investigación Académica

* **Análisis exploratorio:** Identificar patrones en los datos
* **Validación de hipótesis:** Verificar hallazgos experimentales
* **Generación de figuras:** Crear visualizaciones para publicaciones

### F.6.2 Desarrollo de Sistema

* **Debugging:** Identificar problemas en modelos específicos
* **Optimización:** Comparar configuraciones y parámetros
* **Monitoreo:** Tracking de performance a lo largo del tiempo

### F.6.3 Presentaciones y Demos

* **Demos interactivas:** Mostrar capacidades del sistema en tiempo real
* **Presentaciones:** Generar visualizaciones para audiencias técnicas
* **Reportes ejecutivos:** Crear resúmenes para stakeholders no técnicos

## F.7 Métricas de Performance de la Aplicación

### F.7.1 Tiempo de Carga

| Componente | Tiempo Promedio | Optimización |
| --- | --- | --- |
| **Carga inicial** | 2.3 segundos | Caching de datos |
| **Cambio de página** | 0.8 segundos | Session state |
| **Generación de gráficos** | 1.5 segundos | Plotly optimizado |
| **Exportación PDF** | 4.2 segundos | Procesamiento asíncrono |

### F.7.2 Uso de Recursos

* **Memoria RAM:** ~150MB (datos cargados)
* **CPU:** Picos del 20% durante generación de gráficos
* **Storage:** ~50MB cache de visualizaciones
* **Network:** Mínimo (datos locales)

## F.8 Mantenimiento y Actualizaciones

### F.8.1 Actualizaciones de Datos

def update\_data\_sources():  
 """Actualiza fuentes de datos experimentales"""  
   
 # Detectar nuevos archivos de resultados  
 new\_files = scan\_for\_new\_results()  
   
 # Validar formato y completitud  
 validated\_files = validate\_results\_files(new\_files)  
   
 # Actualizar cache de la aplicación  
 update\_app\_cache(validated\_files)  
   
 # Notificar a usuarios activos  
 st.rerun()

### F.8.2 Monitoreo de Performance

def monitor\_app\_performance():  
 """Monitorea performance de la aplicación"""  
   
 metrics = {  
 'load\_time': measure\_load\_time(),  
 'memory\_usage': get\_memory\_usage(),  
 'active\_users': count\_active\_sessions(),  
 'error\_rate': calculate\_error\_rate()  
 }  
   
 # Log métricas  
 logger.info(f"App metrics: {metrics}")  
   
 # Alertas si performance degrada  
 if metrics['load\_time'] > 5.0:  
 send\_performance\_alert(metrics)

## F.9 Conclusión

La aplicación Streamlit proporciona una interfaz comprehensiva para explorar y analizar los resultados experimentales del sistema RAG. Su arquitectura modular permite fácil extensión y mantenimiento, mientras que sus capacidades de visualización facilitan el entendimiento de patrones complejos en los datos experimentales.

### F.9.1 Beneficios Principales

1. **Accesibilidad:** Interfaz web intuitiva sin necesidad de conocimientos técnicos
2. **Interactividad:** Exploración dinámica de resultados experimentales
3. **Reproducibilidad:** Visualizaciones consistentes basadas en datos verificables
4. **Extensibilidad:** Arquitectura modular para agregar nuevas funcionalidades

### F.9.2 Uso Recomendado

* **Análisis exploratorio** de resultados experimentales
* **Validación** de hallazgos de investigación
* **Generación de reportes** para audiencias diversas
* **Desarrollo iterativo** del sistema RAG

**Acceso:** Las aplicaciones están disponibles ejecutando: - streamlit run src/apps/main\_qa\_app.py (aplicación principal) - streamlit run src/apps/cumulative\_metrics\_results\_matplotlib.py (resultados experimentales)

Después de seguir las instrucciones de configuración del Anexo C.