# Enunciado del Proyecto Final Del Curso de Infraestructuras Paralelas y Distribuidas

---

## 1. Motivación

En el contexto actual, las aplicaciones de Machine Learning requieren cada vez más escalabilidad y capacidad de procesamiento en paralelo para atender grandes volúmenes de datos y ofrecer respuesta en tiempo real. Ray es un framework de Python diseñado para paralelizar y distribuir tareas de cómputo de forma sencilla, mientras que la arquitectura de microservicios en la nube permite desplegar y escalar componentes de manera independiente. Este proyecto integra ambos enfoques para que los estudiantes apliquen conceptos de paralelismo y distribución en un problema práctico de ML, desde la paralelización del código hasta el despliegue de microservicios y un cliente que consuma las APIs desarrolladas .

---

## 2. Objetivo del Proyecto

Desarrollar, desplegar y evaluar una aplicación de Machine Learning que:

1. **Paralelice** el procesamiento de datos y el entrenamiento/inferencia con Ray, utilizando decoradores como `@ray.remote` y `@ray.serve`.
2. **Distribuya** la aplicación en microservicios sobre instancias EC2 de AWS, exponiendo funcionalidades clave mediante APIs creadas con alguna librería en Python, FastAPI o Flask.
3. **Implemente** un cliente frontend que consuma dichas APIs para realizar inferencias de forma interactiva.

---

## 3. Descripción de las etapas

1. **Paralelización con Ray**

    - Identificar el núcleo computacional del problema de ML (e.g., identificar la parte más costosa computacional. Hacer la observación del comportamiento de su aplicación (*profiling*) y decidir qué parte del código es candidato a ser paralelizado y/o distribuido).
    - Refactorizar funciones críticas para que sean tareas Ray (`@ray.remote`) y/o endpoints de servicio (`@ray.serve`).
2. **Containerización y despliegue**
    - Empaquetar cada componente (servicios Ray, APIs FastAPI o Flask) en contenedores Docker.
    - Provisionar instancias EC2 en AWS y orquestar el despliegue (puede usarse Docker Compose).
3. **Desarrollo del cliente**
    - Crear una interfaz web o de línea de comandos que invoque las APIs desplegadas y muestre resultados de inferencia.

---

# 4. Entregables

1. **Repositorio de código completo** en GitHub (o equivalente), con
    - Carpetas separadas para la paralelización (Ray), APIs y cliente.
    - `README.md` con instrucciones de instalación, ejecución y despliegue en AWS.
2. **Informe técnico** (mín. 8 páginas) que incluya:
    - Diseño de la arquitectura (diagramas).
    - Detalles de paralelización y distribución.
    - Resultados de rendimiento (benchmarking).
3. **Scripts de despliegue** (e.g. Bash) y/o archivos `docker-compose.yml`.
4. **Cliente frontend** funcionando, con código y guías de uso.
5. **Presentación** (diapositivas + video) de máximo 10 minutos para exponer metodología y resultados.

---

# 5. Rúbrica de Evaluación

| Criterio | Descripción | Peso |
|---|---|---|
| **Diseño arquitectónico y selección de tecnologías** | Claridad y coherencia en la arquitectura propuesta; justificación del uso de Ray, microservicios y AWS EC2. | 20% |

| | | |
|---|---|---|
| **Implementación paralela con Ray** | Uso correcto de `@ray.remote` y `@ray.serve`; eficiencia en la paralelización y correcta gestión de dependencias. | 25% |
| **Implementación de microservicios y APIs** | Contenerización adecuada, despliegue en EC2, diseño REST de las APIs y manejo de errores. | 20% |
| **Benchmarking y análisis de rendimiento** | Comparativa secuencial vs. paralelo vs. distribuido; presentación de métricas, gráficos y conclusiones bien fundamentadas. | 20% |
| **Documentación, calidad de código y presentación final** | Legibilidad del código, claridad del `README.md`, calidad del informe y eficacia de la presentación oral. | 15% |

# 6. Referencias

Proyecto_Paralelas_Distribuidas_Ray_ML