Propuesta de Tesina para la obtención del grado de Licenciado en Ciencias de la Computación

2 de diciembre de 2025

Postulante: Joaquin Arroyo
Legajo: A-4294/3
Director: Matias Gerard
Codirector: Leandro Vignolo

1. Situación del postulante

Al momento de la presentación de esta propuesta, el estudiante tiene aprobadas todas las materias requeridas para la carrera, a la espera de la carga de notas correspondientes a: *Seminario de Práctica Profesional*.

Actualmente trabaja 40 horas semanales y destinará un mínimo de 10 horas semanales al desarrollo de la tesina.

2. Título

Desarrollo de estrategias para el entrenamiento paralelo de modelos neuronales en GPUs individuales.

3. Motivación y objetivo general

El entrenamiento de redes neuronales profundas representa una de las tareas más costosas en términos de tiempo y energía dentro del aprendizaje automático. Si bien las GPU se han consolidado como el hardware dominante para este tipo de cargas, en numerosos escenarios de investigación o validación de modelos la utilización efectiva del dispositivo

suele ser baja. En particular, cuando se entrenan múltiples redes pequeñas o medianas de forma secuencial —como ocurre en la validación cruzada, la búsqueda de hiperparámetros o los métodos evolutivos—, la GPU puede permanecer ociosa durante una proporción significativa del tiempo de ejecución, o también puede que durante el tiempo que se ocupa no se utilicen los recursos al 100 %.

A pesar de que existen estrategias para mejorar la utilización de GPU, muchas de ellas están diseñadas para escenarios *multi-GPU*, *multi-tenant* o entornos *cloud*, y no resultan adecuadas para optimizar la concurrencia dentro de una única GPU estándar. Tecnologías como **MIG** [3] o **vGPU** [4] requieren hardware especializado no disponible en GPUs de uso general, además de configuraciones de particionado o virtualización que limitan su aplicabilidad en entornos académicos con recursos de hardware convencionales. Asimismo, enfoques avanzados como **Salus** [13], **Zico** [7] o **MIGER** [14] introducen mecanismos complejos de preempción fina, coordinación dinámica de memoria o planificación híbrida, lo que implica modificaciones profundas en los frameworks de entrenamiento o la necesidad de infraestructura distribuida.

Estas limitaciones motivan la necesidad de explorar alternativas accesibles y reproducibles que permitan mejorar la utilización del dispositivo dentro de una única GPU, evitando depender de hardware especializado o de reconfiguraciones intrusivas. En este contexto, la presente tesina se propone **analizar y comparar diferentes estrategias de concurrencia de entrenamientos en una misma GPU**, considerando enfoques que actúan en distintos niveles de la pila de software: el nivel *driver*, representado por **CUDA MPS** [2]; el nivel *runtime*, mediante **Ray** [11]; y el nivel *framework*, a través de un enfoque inspirado en **UnifiedNN** [15]. Cada técnica será evaluada experimentalmente en escenarios controlados con el fin de cuantificar su impacto sobre la utilización de la GPU, el rendimiento y la eficiencia energética.

4. Fundamentos y estado del conocimiento

4.1. Concurrencia a nivel driver y runtime

En la capa más baja, la concurrencia puede lograrse a través de mecanismos de compartición de GPU provistos por el propio entorno de ejecución de CUDA. El servicio **CUDA Multi-Process Service (MPS)** [2] permite que múltiples procesos compartan los Streaming Multiprocessors (SMs) de una misma GPU, habilitando la ejecución paralela sin necesidad de reconfigurar el hardware. Estudios previos indican que MPS puede mejorar el throughput entre 1.8x y 3.5x respecto de la ejecución secuencial, con un overhead inferior al 5 % [14]. No obstante, cada proceso mantiene su propio contexto y espacio de memoria, lo cual en determinados contextos puede duplicar datos y limitar

el aprovechamiento de la VRAM en cargas intensivas.

En un nivel superior se encuentra **Ray Tune** [11], un framework de orquestación distribuida que permite ejecutar y monitorear múltiples entrenamientos en paralelo. Ray administra los recursos de GPU asignando fracciones de cada dispositivo (por ejemplo, 0.25 GPU por experimento) y coordinando el ciclo de vida de los procesos, lo que facilita la ejecución de experimentos concurrentes con trazabilidad y reproducibilidad. Sin embargo, esta asignación fraccionada es **estática** y debe fijarse antes del inicio del entrenamiento, lo que limita la posibilidad de ajustar dinámicamente el uso de GPU según la carga real del modelo. A diferencia de MPS, Ray no controla la GPU a nivel de kernel o memoria, pero sí ofrece mecanismos de pausa, reanudación y reasignación de recursos mediante *schedulers* como ASHA [10] o PBT [5].

Además de estas herramientas, trabajos como **Salus** [13] y **Zico** [7] han introducido mecanismos de planificación fina y coordinación de memoria entre procesos concurrentes, demostrando que el manejo dinámico de recursos puede mejorar significativamente la utilización global del hardware. En esta misma línea, **PipeSwitch** [6] propone una estrategia de conmutación rápida de modelos basada en pipelines, que reduce el overhead asociado al cambio de contexto entre redes neuronales y ofrece una alternativa interesante para la ejecución de múltiples modelos independientes. Aunque dichos enfoques exceden el alcance de esta propuesta, sus resultados fundamentan la necesidad de explorar estrategias de concurrencia eficientes dentro de una GPU única.

4.2. Unificación de modelos a nivel framework

En el nivel de framework, distintos enfoques proponen reducir el overhead de múltiples entrenamientos fusionando modelos dentro de un mismo grafo computacional. **HFTA** (Horizontally Fused Training Array) [1] propone la fusión horizontal de redes homogéneas para acelerar tareas de validación cruzada e hiperparámetros. Más recientemente, **UnifiedNN** [15] presenta un marco general que permite ejecutar múltiples modelos heterogéneos de forma unificada, compartiendo operadores y memoria intermedia. Esta línea de trabajo, aún emergente, muestra un alto potencial de mejora en la eficiencia del entrenamiento y en la reducción del consumo energético. En esta tesina se implementará un **enfoque basado en UnifiedNN**, adaptado al contexto de una GPU individual.

4.3. Eficiencia energética y criterios de evaluación

La eficiencia energética se ha consolidado como un eje fundamental en la evaluación de sistemas de inteligencia artificial. El benchmark **MLPerf Power** [12] establece un marco metodológico para medir consumo de potencia y energía durante el entrenamiento de

modelos, permitiendo comparar configuraciones de hardware y software bajo métricas reproducibles. El presente trabajo tomará como referencia esta metodología, complementada con mediciones de utilización y memoria obtenidas mediante las bibliotecas NVML de NVIDIA, a fin de analizar de manera integral el rendimiento y la eficiencia energética de cada enfoque.

5. Objetivos específicos

1. Revisar el estado del arte sobre técnicas de entrenamiento concurrente en GPU única, considerando enfoques a nivel driver, runtime y framework.
2. Definir escenarios experimentales representativos (datasets, modelos y tipos de entrenamiento) que permitan observar patrones de subutilización de GPU.
3. Establecer una línea base de ejecución concurrente utilizando **CUDA MPS**.
4. Establecer una línea base de ejecución concurrente mediante **Ray**.
5. Desarrollar e implementar un enfoque basado en los principios de **UnifiedNN** que permita la ejecución unificada de múltiples modelos dentro de un mismo proceso.
6. Comparar los tres enfoques en términos de throughput, utilización de GPU, consumo de memoria y energía, y eficiencia global.

6. Metodología y plan de trabajo

El plan de trabajo se desarrollará siguiendo los objetivos específicos planteados, integrando componentes de análisis bibliográfico, experimentación y desarrollo.

En primer lugar se llevará a cabo una revisión exhaustiva del estado del arte, abordando mecanismos de compartición de GPU como **CUDA Multi-Process Service (MPS)** [2, 14], frameworks de orquestación de entrenamientos como **Ray Tune** [11], y estrategias de fusión o unificación de modelos propuestas en trabajos recientes, entre ellas **HFTA** [1] y **UnifiedNN** [15]. También se considerarán estudios y metodologías de evaluación energética, particularmente el estándar **MLPerf Power** [12], con el fin de establecer criterios de medición comparables.

A partir de esta revisión se definirán los escenarios experimentales, seleccionando datasets de referencia como *MNIST* [9], *CIFAR-10* [8] y otros conjuntos similares de libre acceso y ampliamente utilizados en la literatura, junto con modelos representativos de distinta complejidad: perceptrones multicapa (MLP), redes convolucionales y arquitecturas *ResNet*.

Asimismo, se considerarán configuraciones de entrenamiento tales como validación cruzada (*k-fold*) o búsqueda de hiperparámetros. Si bien estas técnicas no implican necesariamente una baja utilización de GPU, suelen implementarse de manera secuencial ejecutando múltiples modelos de forma independiente, lo que en la práctica conduce a períodos prolongados de ociosidad del dispositivo. Este tipo de patrones motivan la exploración de estrategias de concurrencia que permitan aprovechar mejor la capacidad de cómputo disponible.

Posteriormente se establecerán líneas base de ejecución concurrente utilizando, por un lado, **CUDA MPS**, para evaluar la concurrencia a nivel de *driver*, y por otro, **Ray**, actuando como orquestador a nivel *runtime*. En ambos casos se registrarán métricas de rendimiento, utilización de GPU, memoria y energía.

Finalmente, se desarrollará un enfoque basado en los principios de **UnifiedNN**, adaptado al contexto de una GPU individual, que permitirá la ejecución conjunta de múltiples modelos dentro de un mismo proceso. Este desarrollo representará el enfoque a nivel de *framework*, posibilitando una comparación integral entre los tres niveles de la pila de software: driver, runtime y framework.

6.1. Cronograma de trabajo

El cronograma tentativo de la tesina es el siguiente:

Actividad	**Meses**											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Revisión del estado del arte												
Definición de escenarios experimentales												
Implementación y análisis de la línea base con MPS												
Implementación y análisis de la línea base con Ray												
Desarrollo e implementación del enfoque basado en UnifiedNN												
Evaluación comparativa												
Redacción del informe final												

Referencias

[1] Daniel Bauer et al. Hfta: Horizontally fused training array for efficient hyperparameter optimization. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[2] NVIDIA Corporation. Cuda multi-process service (mps) overview. `https://docs.nvidia.com/deploy/mps/index.html`, 2023. Accedido: noviembre de 2025.

[3] NVIDIA Corporation. Nvidia multi-instance gpu user guide. `https://docs.nvidia.com/datacenter/tesla/mig-user-guide/`, 2023. Accedido: noviembre de 2025.

[4] NVIDIA Corporation. Nvidia virtual gpu (vgpu) software overview. `https://docs.nvidia.com/grid/`, 2023. Accedido: noviembre de 2025.

[5] Max Jaderberg et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[6] Zhen Jia et al. Pipeswitch: Fast pipeline-based model switching for dnn applications. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021.

[7] Zhen Jia et al. Zico: Gpu memory sharing for concurrent dnn training. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021.

[8] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[11] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[12] Vijay Janapa Reddi et al. Mlperf power: A benchmark for energy efficiency in machine learning. In *Proceedings of the IEEE/MLPerf*, 2020.

[13] Guanhua Yu et al. Salus: Fine-grained gpu sharing primitives for deep learning applications. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2019.

[14] Tian Zhang et al. Miger: Multi-instance gpu + mps hybrid scheduling for deep learning workloads. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.

[15] Wei Zhang et al. Unifiednn: Unified model training for heterogeneous neural networks. *arXiv preprint arXiv:2306.10564*, 2023.