This is an excellent choice for a project, as it directly addresses a critical and complex aspect of high-performance computing on GPUs. Optimizing a CUDA kernel's launch configuration—specifically the **thread block size** and **grid dimensions**—is a well-known challenge. A deep learning approach offers a powerful way to automate this process, which is currently done through manual trial-and-error or heuristic-based methods.
Here is a detailed breakdown of the project, expanding on your initial idea into a full-fledged research and development plan.

Project Title: An Intelligent CUDA Kernel Optimizer Using Deep Learning

1. Problem Statement

The performance of a GPU kernel is highly sensitive to its launch configuration, which includes the number of threads per block and the total number of blocks in the grid. An under-sized block may fail to hide memory latency, while an over-sized block may lead to resource contention. The optimal configuration is non-trivial and depends on the specific kernel's characteristics (e.g., arithmetic intensity, memory access patterns), the GPU's microarchitecture (e.g., number of Streaming Multiprocessors - SMs, shared memory size), and the input data size. The goal of this project is to build a deep learning model that can predict the optimal launch configuration to maximize kernel throughput, minimizing the need for manual tuning.

2. Proposed Methodology

This project can be framed as a supervised learning problem. The model will be trained on a dataset of kernels, their various launch configurations, and the resulting performance metrics.
A. Data Collection and Feature Engineering
This is the most critical and labor-intensive part of the project. A robust dataset is key to the model's success.

- **Kernels**: Select a diverse set of GPU kernels from benchmarks like Rodinia, Parboil, or CUDA's own samples. This ensures the model learns to generalize across different types of workloads (e.g., data-parallel, memory-bound, compute-bound).
- **Input Features (x)**: These features will be extracted from each kernel and its execution environment to serve as input to the deep learning model.
    - **Static Features**: Program-level characteristics extracted from the kernel's source code or its compiled binary. Examples include:
        - Number of instructions (total, arithmetic, memory).

        - Number of registers used per thread.
        - Amount of shared memory used per block.
        - Control flow complexity (e.g., number of branches).
    - **Dynamic Features**: Microarchitectural counters collected during a short profiling run. These provide insights into the kernel's runtime behavior.
        - Memory access patterns (e.g., memory transaction coalescing, global memory access rate).
        - Instruction throughput (e.g., compute-to-memory ratio, arithmetic intensity).
        - Cache hit rates.
    - **System Features**: Characteristics of the GPU hardware.
        - Number of Streaming Multiprocessors (SMs).
        - L1/L2 cache sizes.
        - Max threads per SM, max shared memory per SM.

B. Model Architecture

A deep neural network is a suitable choice for this task. The model's architecture should be designed to handle both numerical and potentially categorical features.

- **Model Type**: A multi-layered perceptron (MLP) or a deeper feed-forward neural network is a strong starting point. The input layer will take the features from Step 2A.
- **Output Layer (y)**: The model's output will be the predicted optimal launch configuration. This can be modeled in a few ways:
    - **Regression**: Predict a tuple of (threads_per_block, grid_size) as continuous values.
    - **Classification**: Pre-define a set of common launch configurations (e.g., 64, 128, 256, 512 threads per block) and have the model classify the best one. The grid size could then be calculated based on the data size. This may be more practical for implementation.

C. Training and Evaluation

- **Training**: Train the model on the collected dataset. The objective function will be to minimize the difference between the predicted performance and the actual observed performance (e.g., kernel_runtime). The loss could be Mean Squared Error (MSE) for a regression model or cross-entropy for a classification model.
- **Evaluation**: Use a hold-out test set of kernels not seen during training. The primary metric will be the model's accuracy in predicting the fastest configuration. A good metric would be "Top-K Accuracy", where the model's prediction is considered correct if it falls within the top K best-performing configurations. Compare the model's performance to a simple heuristic (e.g., a fixed block size of 256) and brute-force search.

3. Expanded Project Ideas & Challenges

This project can be expanded into several research directions:

- **Online Reinforcement Learning (RL) Optimizer**: Instead of a supervised model that predicts a configuration, an RL agent could learn to make sequential decisions about resource allocation. The agent could observe the current kernel's execution state (from performance counters) and make real-time adjustments to the launch configuration. The reward signal would be the improvement in kernel performance. This would be a more complex but potentially more powerful solution for dynamic workloads.
- **Model Generalization**: The biggest challenge is ensuring the model generalizes to new kernels and unseen GPU architectures. One way to approach this is to use **transfer learning**, where a model trained on one GPU is fine-tuned for a new, unseen GPU, or trained on a set of kernels and then applied to a new one.
- **Interpretable AI**: Investigate which features the model finds most important. For instance, using techniques like SHAP or LIME, you could determine if the model is correctly identifying that register pressure or shared memory usage are the primary bottlenecks for a given kernel. This would provide valuable insights for compiler and hardware designers.

This project offers a blend of practical engineering and cutting-edge research, providing a strong foundation for a thesis or a robust development project.