Universitat Autònoma de Barcelona
Degree in Artificial Intelligence
Supervisor: Vanessa Moreno Font

OCR Implementation in a Jetson Nano

Pol Medina Arévalo

May 18, 2025

ABSTRACT

This project investigates the deployment of optimized Optical Character Recognition (OCR) models on the NVIDIA Jetson Nano T1 for real-time text extraction in industrial settings. We aim to address the critical challenge of balancing accuracy and inference speed on resource-constrained edge devices. Specifically, we explore the adaptation and optimization of deep learning-based OCR architectures for efficient execution on the Jetson Nano. Our methodology involves a multi-stage process: (1) establishing baseline performance on a high-performance server, (2) applying model optimization techniques such as quantisation, pruning, and TensorRT acceleration to reduce inference time, and (3) deploying these optimized models on the Jetson Nano. We will evaluate performance using metrics such as character recognition accuracy and inference latency, comparing the results against unoptimized models. The expected contributions include a benchmark of OCR model performance on the Jetson Nano, demonstrating the feasibility of achieving real-time processing within industrial constraints, and a reproducible methodology for optimizing deep learning models for edge deployment. This research contributes to the advancement of embedded AI for industrial automation, providing insights into the trade-offs between accuracy and efficiency in real-time OCR applications.

1 Problem definition

The primary goal of this project is to implement optical character recognition (OCR) models, which involve converting different types of images into machine-readable text, powered by artificial intelligence (AI) on an NVIDIA Jetson Nano T1 microprocessor. The Jetson Nano is designed specifically for AI applications, and due to its powerful GPU and AI-optimized architecture is well-suited to this task. This leverages its capability for efficient parallel processing, making it an ideal platform for running OCR tasks and offering a compact solution for edge processing where small form factors and portability are essential.

Within this goal there is a mandatory requirement, which involves using deep learning models to perform OCR while optimizing them for speed and efficiency. The challenge is to ensure that these models not only perform well but also have extremely low inference times. This means the project explores AI model acceleration techniques to reduce inference time while maintaining accuracy. The Jetson Nano's GPU, combined with various optimization methods, will make it possible to achieve highly efficient OCR performance that is suitable for the industrial environment. Further explanations in metrics and specific objectives, as well as optimization techniques will be explained in further sections.

The specific use case for this project is based in industrial settings, particularly in manufacturing or warehouse environments where product codes and bar codes are scanned. Codes will consist of printed strings of numbers and letters like the one in image 1. The OCR models will be used to interpret these codes at high speed for further use according to the needs of the processes. In these industries, efficiency is crucial. Thousands of products may need to be processed in a very short amount of time, making it necessary to deploy a system that can handle this task swiftly. The Jetson Nano's processor, thanks to its portability and AI-optimized capabilities, is the perfect choice for this type of application enabling real-time processing, essential in these environments.

1.1 Relevant hardware specifications

For the development of the project two different platforms will be used: a server with available GPUs and the Jetson Nano. The code will be first developed in the cluster's GPUs and it will be then transferred to the Jetson Nano environment with the corresponding compatibilities finetuning. The specifications of two available GPUs and Jetson are specified in table 1.

It is relevant to understand that NVIDIA Jetson Nano works with JetPack [10], which is a software development kit (SDK) for NVIDIA Jetson devices. It provides everything needed to develop AI and computer vision applications, including Ubuntu based OS, CUDA, cuDNN and TensorRT [17]. It offers an optimized software stack aimed for edge computing on these devices.

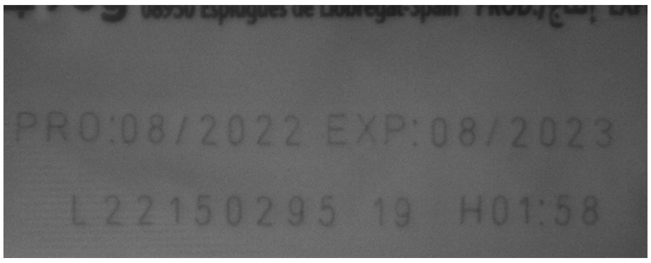


Figure 1: Example of case use images with codes

	Performance	*Cuda Cores*	*Video Memory*	*Power Consumption*
Jetson Nano T1	∼0.5 TFLOPS	128	4GB LPDDR4	5-10W
RTX 2080 Ti	∼13.45 TFLOPS	4,352	11 GB GDDR6	250W
RTX 3090	∼35.6 TFLOPS	10,496	24 GB GDDR6	350W

Table 1: Hardware specifications for available resources [9]

2 Previous Context and State of the Art

In the realm of Optical Character Recognition (OCR), achieving a balance between accuracy and computational efficiency is crucial, particularly when deploying models on resource-constrained devices like the NVIDIA Jetson Nano. While cloud-based OCR solutions leverage large Transformer-based architectures to attain high accuracy, these models pose significant challenges for edge deployment due to their high computational and memory requirements. Moreover, the use of cloud-based solutions could increase the cost and would need constant connection to the servers. These are not suitable when looking for cost-effective solutions.

2.1 Industrial Applications of Jetson Nano in OCR and Vision Tasks

The NVIDIA Jetson Nano has been effectively utilized in various industrial settings for OCR and vision-related tasks, demonstrating its capability in real-world applications:

- **License Plate Recognition:** Implementations have been developed for real-time license plate detection and recognition using the Jetson Nano [14], showcasing its ability to handle OCR tasks efficiently in edge computing scenarios.
- **Warehouse Automation:** Embedded camera solutions powered by Jetson Nano [15] have been integrated into conveyor systems for OCR applications, enhancing automation processes in warehouse environments.
- **Smart Surveillance:** Devices like the SmartCam T1023 [16], which utilize the Jetson Nano, are deployed for tasks including parking detection and surveillance, leveraging on-device OCR capabilities to process and analyze visual data at the edge.

These examples underscore the Jetson Nano's versatility and effectiveness in deploying OCR and vision applications within industrial contexts, particularly where real-time processing and cost-efficiency are paramount.

2.2 Benchmarks of Existing OCR Models

Recent evaluations highlight the trade-offs between accuracy and inference time among various OCR models, particularly in terms of efficiency for edge computing:

EasyOCR: An open-source model achieving a character accuracy of approximately 85% with an inference time of 0.042 seconds per image. EasyOCR [1] utilizes a CRNN architecture, balancing simplicity and performance.

PaddleOCR: Another open-source solution with a character accuracy of around 90% and an inference time of 0.110 seconds per image. PaddleOCR [2] offers a range of models, including those optimized for mobile devices, demonstrating a focus on efficiency.

Tesseract OCR: A traditional OCR engine, efficient for printed text but struggles with complex layouts and handwritten text. It offers speed but lacks the deep learning-based accuracy of newer models [4].

TrOCR: A Transformer-based OCR model achieving over 96% word-level accuracy on structured document tasks and high performance on handwritten text (e.g., CER $\approx$ 3% on IAM). While accurate, inference time can be high on edge devices unless optimized [11].

PP-OCRv2: An improved version of PaddleOCR designed for mobile deployment. It offers better accuracy than its predecessor with small model size (∼3.5MB) and fast inference, making it ideal for edge computing [12].

SVTRv2: A lightweight OCR architecture using CTC decoding that outperforms encoder-decoder models in both speed and accuracy, especially effective in scene text recognition [13].

Google Cloud Vision API: A cloud-based service offering a character accuracy of about 95% with an inference time of 0.063 seconds per image. While highly accurate, cloud-based APIs introduce latency and are not ideal for real-time edge processing [3].

These metrics underscore that while cloud-based models like Google Cloud Vision provide high accuracy, they may not be suitable for real-time applications on edge devices due to latency and resource constraints. Conventional engines like Tesseract are fast but less capable with complex layouts or handwritten text. In contrast, newer transformer and lightweight models such as TrOCR, PP-OCRv2, and SVTRv2 offer strong performance for edge deployment, particularly when combined with model compression or quantization techniques.

2.3 Challenges with Transformer-Based Models on Edge Devices

Modern OCR systems increasingly rely on Transformer-based architectures, which, while highly effective, present significant challenges for deployment on resource-constrained hardware such as the Jetson Nano:

- **High Memory Usage:** Self-attention mechanisms require large memory overhead, making real-time inference difficult on low-power devices. [5]
- **Inference Latency:** Unlike CNN-based models, Transformer-based architectures often introduce longer inference times due to the sequential nature of attention computations. Techniques like distillation are being explored to mitigate this. [6]
- **Computational Demands:** Transformers require substantial parallel processing power, making them inefficient for devices with limited GPU resources. Model quantization and pruning are essential for deploying these models on edge devices. [7]

While cloud-based OCR systems employing Transformer architectures offer high accuracy, their computational demands render them impractical for real-time applications on edge devices. Therefore, developing lightweight OCR models optimized for edge computing—through techniques such as model quantization, pruning, and TensorRT acceleration—presents a viable solution for deploying efficient OCR systems in resource-constrained environments like our industrial use case setting.

Moreover, regarding directly to the Jetson Nano, while there are existing projects handling several computer vision tasks such as object segmentation or human pose estimation, there does not exist a direct approach to accelerated OCR computation in the platform. Works such as *On-device real-time hand gesture recognition with deep learning* [8] show the capability of Jetson Nano for computer vision, but lack the OCR focus. This creates the necessity of creating this implementation in a Jetson device.

Despite all mentioned above, there exists a transformer based OCR called *TrOCR* that has been published by Microsoft in different variants called "large", "base" and "small", which can be suitable for edge devices like Jetson Nano.

3 Objectives and Contributions

The primary objective of this project is to develop and evaluate an optimized Optical Character Recognition (OCR) system for real-time text extraction on the NVIDIA Jetson Nano T1, tailored for industrial use cases. The core contributions will include:

Optimizing AI models for Jetson Nano: achieve a short inference time similar to what is expected from baseline models (e.g. 50 milliseconds per image with EasyOCR) on the Jetson Nano. To do this we will employ techniques such as model quantization (8-bit and mixed-precision), pruning (magnitude-based and structured), and TensorRT acceleration. The baseline model, trained on a server with RTX 3090 would, in a desired way, be optimized to reduce its size considerably while maintaining a character recognition accuracy similar to the original model. This will be done with TensorRT model transformation.

Benchmarking the efficiency: We will conduct a comparative analysis on OCR models (EasyOCR, PaddleOCR and small TrOCR as main architectures), evaluating their performance in terms of inference time, character recognition accuracy, and model size. This benchmark will provide insights into the trade-offs between these metrics on both server and Jetson Nano environments.

Adapting the models to the Jetson Nano: All selected models will be executed as baseline and then converted to TensorRT format and optimized for efficient execution in the Jetson Nano. This adaptation will involve optimizing layers and model parameters to maximize the GPU utilization of the Jetson Nano. The same parameters and configuration will be ran through both the server and the Jetson. In order to adapt the OCRs to the device, different environments will be Dockerized. It is also expected to compute the benchmarking with execution inside containers and outside them to compare computational differences.

Evaluation of final performance: A comprehensive evaluation will be conducted, comparing the inference time and character recognition accuracy of the unoptimized models, the optimized models running on the server, and the final models deployed on the Jetson Nano. The optimized version is aimed to achieve an undetermined minimum speed-up in the inference time in the Jetson Nano compared to the baseline while maintaining a similar character recognition accuracy (ranging within 5% to 10% difference) in the industrial use case dataset.

The main contributions of this work include:

- Developing an optimized OCR system tailored for real-time edge computing on the Jetson Nano, demonstrating the feasibility of achieving high performance within the constraints of resource-limited hardware.
- Providing a detailed benchmark of inference speed, character recognition accuracy, and model size for various OCR models in embedded environments, offering valuable insights for future research and development.
- Demonstrating and evaluating hardware acceleration techniques, specifically TensorRT optimization, for real-time OCR in industrial applications, showcasing the potential for efficient edge deployment.
- Providing a reproducible methodology for optimizing deep learning models for edge deployment, including code and documentation for further similar applications.

4 Methodology and Work Plan

The project will follow a structured methodology to achieve its objectives. The work is divided into several stages:

Environment Setup on the Server: We will establish a high-performance server environment equipped with an NVIDIA RTX 3090 and an NVIDIA RTX 2080 Ti. The software stack will include Ubuntu, CUDA, cuDNN, PyTorch and OpenCV with CUDA support. This environment will facilitate efficient training and initial testing of OCR models.

Image Preprocessing: To enhance OCR performance, we will apply a series of different preprocessing techniques using OpenCV with CUDA acceleration. This can include:

- Noise reduction using Gaussian blur and median filtering.
- Adaptive thresholding (e.g., Otsu's method) to binarize the images and isolate characters.

Model Selection: We will evaluate many OCR models, as long as resource constraints allows us to, including:

- EasyOCR, utilizing a CRNN (Convolutional Recurrent Neural Network) architecture.
- PaddleOCR, exploring their models optimized for mobile and edge devices.
- Small TrOCR, based in a transformer architecture with a reduced size expected to be used in edge devices.

Baseline Model Development: We will use pretrained baseline models on the server using the industrial dataset, which consists of images containing alphanumeric codes similar to the example shown in Figure 1.

Model Optimization: To reduce model size and improve inference speed, we will apply the following techniques:

- Quantization: n-bit and mixed-precision quantization using TensorRT.
- Pruning: Magnitude-based and structured pruning to remove redundant weights and connections.

Jetson Nano Setup and Model Adaptation: Models will be deployed on the Jetson Nano and converted to TensorRT format for optimization. The JetPack SDK will be utilized to configure the Jetson Nano docker platform and different containers will be created for the proper execution of the models.

Performance Evaluation: We will evaluate the performance of the models using the following metrics:

- Character Recognition Accuracy: Measured as the percentage of correctly recognized characters.
- Inference Time: Measured in milliseconds per image, to assess real-time processing capabilities.
- Model Size: Measured in megabytes and number of parameters, to evaluate the efficiency of optimization techniques.

We will compare the performance of the baseline models, the optimized models on the server, and the final models on the Jetson Nano (also baselines and optimized) in CPU and GPU computation.

Throughout the project, iterative refinements will be made based on performance evaluations to further optimize the models. The ultimate goal is to develop an OCR system that meets industrial requirements while operating efficiently on resource-constrained hardware.

4.1 Work plan over weeks

Phase/Task	***Weeks***
Phase 1: Setup & Preparation	**(Weeks 1-4)**
- Server environment setup (CUDA, cuDNN, PyTorch, OpenCV...)	Week 1
- Dataset preparation and augmentation (random rotations, scaling, noise addition)	Week 2
- Implementation of image preprocessing pipeline (Gaussian blur, adaptive thresholding...)	Weeks 3-4
Phase 2: Model Selection & Baseline Development	**(Weeks 5-8)**
- Research and selection of OCR models (EasyOCR, PaddleOCR, Transformers...)	Week 5
- Implementation of baseline models on the server	Weeks 6-7
- Evaluation of baseline models (character recognition accuracy, inference time)	Week 8
Phase 3: Jetson Nano Deployment & Adaptation	**(Weeks 9-14)**
- Jetson Nano environment setup (JetPack SDK, TensorRT installation)	Week 9
- Containers creation for Jetson Nano.	Week 10
- Model adaptation for Jetson Nano	Weeks 11-13
- Initial testing and debugging on Jetson Nano	Week 14
Phase 4: Model Optimization	**(Weeks 15-17)**
- Model conversion to TensorRT format	Weeks 15
- Testing of quantization and pruning	Weeks 15
- TensorRT models' adaptation on Jetson Nano	Week 16-17
- Performance metrics extraction on server and Jetson Nano	Week 17
Phase 5: Performance Evaluation & Finalization	**(Weeks 18-20)**
- Comprehensive performance evaluation on Jetson Nano (accuracy, inference time, model size)	Weeks 18-19
- Comparison of Jetson Nano performance w.r.t. server baseline	Week 20
- Writing final report and documentation	Week 20

5 Objectives and Workflow Modifications

5.1 Updated Objectives

During the development phase, some objectives had to be revised. This section outlines the modifications and explains the reasons behind them.

5.1.1 Selected Models and Techniques

1. Currently, only EasyOCR and PaddleOCR are considered for the benchmark study. As observed in preliminary results, Transformer-based models are unfeasible for efficient computation on a Jetson Nano, especially when used without fine-tuning. Custom CRNNs have also been discarded, as they cannot handle code detection tasks effectively without additional training. Since fine-tuning is out of the scope of this project, both Transformer-based models and custom CRNNs are excluded.
2. With regard to AI acceleration techniques, knowledge distillation has also been discarded. Like the previously mentioned models, distillation requires training a compact model using a larger teacher model, which again falls outside the project's scope.
3. In order to convert models to TensorRT, they may first need to be exported to ONNX format. If possible, pruning will be applied before this conversion step. TensorRT will be used to perform post-training quantization, both on the GPU server and the Jetson Nano.
4. Although no specific Transformer model was initially mentioned, Microsoft's TrOCR was selected for evaluation due to its accessibility and availability in multiple sizes suited for edge devices. However, as stated earlier, it has been discarded due to its inefficiency on Jetson Nano.
5. A new approach will be used for the Jetson Nano implementation. While not previously mentioned, all models will be executed within Docker containers. This enhances reproducibility and portability. Additionally, comparing the performance of models running natively on the Jetson OS versus inside Docker containers has been added as a potential task, in order to better understand how the device should be configured for optimal performance.
6. The previously expected model optimization target of reducing inference time to 100 milliseconds has been removed due to the lack of baseline measurements. Instead, a tolerance of 5% to 10% accuracy loss has been set as an acceptable margin for optimized models, acknowledging the uncertainty in expected performance.
7. Finally, inference time will be decomposed into separate stages. While the full process is currently known to last approximately 30 ms on average, this breakdown will help attribute time to specific steps: image preprocessing, text localization, and text recognition.

5.2 Workplan Modification

Regarding the workplan, phases 3 and 4 have been rescheduled. Phase 4, which includes Jetson Nano deployment and adaptation, will now precede the model optimization phase. This change was made to provide a larger time buffer because of compatibility or CUDA-related issues, which are common. Containerization on the Jetson Nano has been added as a subtask in this phase, and the TensorRT optimization process has been moved to the model optimization phase.

Original Plan	**Modified Plan**
Evaluate CRNN, Transformer, and EasyOCR	Only PaddleOCR and EasyOCR considered feasible
Include knowledge distillation	Discarded due to training requirements
Jetson deployment after optimization	Jetson setup happens before optimization
No containerization	Docker-based Jetson deployment added
Fixed optimization goals (100 ms, 5%)	Only relative targets, not absolute, due to unknowns

6 State of project development

Until now, the workplan has been more or less been followed. At week 14, tasks from week 11-13 are being developed (Model adaptation and optimization for Jetson Nano). This means that there are 2 weeks of delay in the development of the project.

6.1 Done tasks by week 14

6.1.1 Phase 1: Setup & Preparation

The server environment is correctly set up and is able to execute the baseline models scripts with CUDA and cuDNN support both in PyTorch and OpenCV. The dataset images have been cropped to only display the target code and nothing else and has been stored in a directory. The models will not have to detect the bounding box for the text but just compute the preprocessing pipeline and the OCR over the images. The preprocessing pipeline has been correctly defined and returns the thresholded input image with the codes as seen in figure 2.

6.1.2 Phase 2: Model selection & baseline development

In the end, EasyOCR and PaddleOCR will be used for optimizing and adapting it to Jetson Nano. Further explanation on this below. Their baseline implementation (without optimization) is already prepared for execution in the cluster as well as the scripts that handle the model logs. They have been evaluated and their preliminar results can be seen in section 7.

6.1.3 Phase 3: Jetson Nano deployment & adaptation

The Jetson Nano has been correctly set up and its environment is ready to run Python 3.6.9 with PyTorch and CUDA support. Containers are also usable and several containers have been created for isolated execution of each different model, baselines and optimized models. Up until here everything is properly working. From this phase they are still remaining the model adaptation and the testing and debugging on Jetson Nano.

6.2 Undone tasks

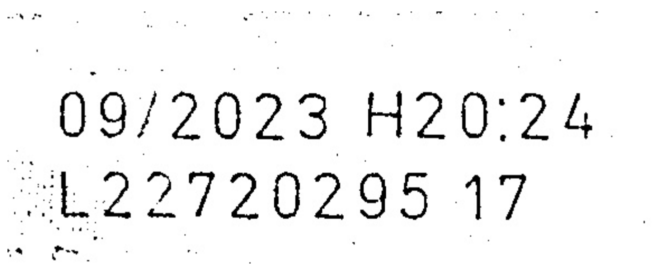
From the model adaptation task forward no further task has been developed. After the alteration between phases 3 and 4, the model optimization has been left aside and the efforts are now focused in the Jetson Nano adaptation. This is due to several obstacles found regarding JetPack compatibilities and other major issues regarding the Jetson Nano environment.

6.2.1 Obstacles founds during project development

The main issue regarding the implementation on the Jetson Nano T1 is the vast amount of incompatibilities that can be found when trying to execute different scripts with packages with different versions. While the server environment has CUDA 12.5 support and Python 3.10.14 the Jetson Nano has CUDA 10.2 support and Python 3.6.9. Moreover, PyTorch versions are limited until 0.13.0 and other versions do not have CUDA support. The problem cannot be solved by using containers, since downloading other more advanced versions still do not ensure the support for CUDA. Since code has not been executed yet, problems with hardware limitations like memory are yet to be found.

This means that in order to adapt the code made in the cluster into the JetPack environment, either different versions have to be tested to find some compatibility that allows the execution of the code or the code must be re written with available and equivalent packages to fulfill the same task. In case of PaddleOCR, for example, needs PaddlePaddle to work, which is the underlying motor of PaddleOCR. This adds a layer of complexity to the installation and use of the package. EasyOCR also demands several package dependencies that are automatically downloaded based on the latest version. These automatic installs harm the Python environment and adds difficulties to the proper adaptation of code into the Jetson Nano environment, either using containers or not.

In summary, the project's progression has been affected by critical software and hardware constraints in the Jetson Nano environment. These include outdated CUDA and Python versions, limited PyTorch support, and complex dependencies of OCR frameworks like PaddleOCR and EasyOCR. As a result, all model optimization efforts have been temporarily paused, and current development is centered on overcoming these adaptation challenges to ensure functional deployment on the Jetson Nano.



09/2023 H20:24
L22720295 17

Figure 2: OCR preprocessed image input example

7 Preliminar results

To display the results, the following tables will be filled up with the performance details on each platform and configuration.

7.1 Baseline models

		Baseline models (PyTorch)			
		Server CPU	**Jetson Nano T1 CPU**	**Server GPU**	**Jetson Nano T1 GPU**
PaddleOCR	**Strict Accuracy**	0.00%		2.18%	
	Character Error Rate (CER)	16.55%		11.46%	
	Character-Level Precision	86.39%		91.22%	
	Character-Level Recall	93.54%		95.64%	
	Total time (1696 samples)	162.02 s		47.34 s	
	Time per sample (ms)	95 ms		28 ms	

		Baseline models (PyTorch)			
		Server CPU	**Jetson Nano T1 CPU**	**Server GPU**	**Jetson Nano T1 GPU**
EasyOCR	**Strict Accuracy**	0.39%		0.60%	
	Character Error Rate (CER)	18.91		27.78%	
	Character-Level Precision	85.51%		81.15%	
	Character-Level Recall	94.31%		84.27%	
	Total time (1696 samples)	322.99 s		74.52 s	
	Time per sample (ms)	190 ms		44 ms	

		Baseline models (PyTorch)			
		Server CPU	**Jetson Nano T1 CPU**	**Server GPU**	**Jetson Nano T1 GPU**
TrOCR	**Strict Accuracy**	0.00%		0.00%	
	Character Error Rate (CER)	70.56%		72.59%	
	Character-Level Precision	68.73%		69.10%	
	Character-Level Recall	41.02%		42.40%	
	Total time (1696 samples)	253.46 s		109.97 s	
	Time per sample (ms)	149 ms		65 ms	

7.1.1 Analysis of baseline results

The baseline results clearly illustrate a crucial trade-off between model complexity and practical deployment feasibility on edge devices. Among the three tested models, PaddleOCR, EasyOCR, and TrOCR, is PaddleOCR which stands out as the most balanced and efficient candidate in terms of both accuracy and inference time. *Notice that the results obtained by computing the predictions with CPU are added to the table, although we will be focusing on GPU results.*

PaddleOCR achieves a significantly higher strict accuracy (2.18%) and a notably lower Character Error Rate (11.46%) than the alternatives, with a respectable inference time of 28 ms per image. EasyOCR, while simpler to implement, loses values in accuracy (Strict Accuracy of 0.60% and CER of 27.78%), showing reduced reliability in code recognition. However, its speed (44 ms per sample) keeps it viable for lightweight applications or scenarios where moderate recognition quality is tolerable.

On the other hand, TrOCR, the transformer-based model, performs substantially worse in this edge context. With 0.00% strict accuracy, a high CER of 72.59%, and the longest inference time (65 ms/sample), TrOCR proves to be unsuitable for the target use case. These limitations can be explained from two key factors:

- **Image Format:** TrOCR underperforms due to its training on datasets with single-line text, while the test data includes codes split across two lines. This structural mismatch severely degrades recognition performance. The model should be finetuned to be able to work with the desired images structure.
- **Architectural Overhead:** Transformer models like TrOCR are parameter-heavy and optimized for high-resource environments. Their computational and memory demands make them ill-suited for real-time processing on Jetson Nano without significant quantization or distillation efforts. Moreover, the download of the models removes the encoder input weights making mandatory a finetuning of the model.

In conclusion, these preliminary results support discarding transformer-based OCR models (TrOCR in this case) from further testing on edge platforms like the Jetson Nano. Instead, emphasis should be placed on further optimizing PaddleOCR through TensorRT conversion, pruning, or quantization to meet the real-time constraints of industrial OCR applications. These findings are crucial for the project's goal of deploying efficient OCR in industrial environments, as they demonstrate that only lightweight models like PaddleOCR are viable for real-time processing on the Jetson Nano, while transformer-based models are not practical given current hardware constraints.

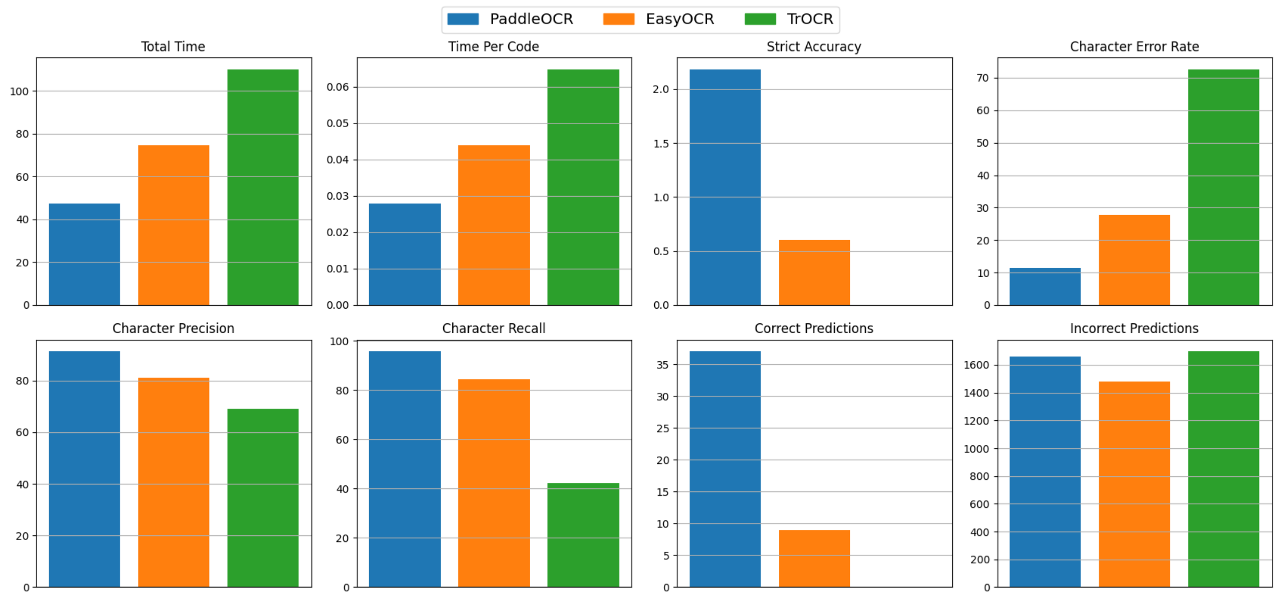


Figure 3: Bar plot with GPU results comparison between TrOCR, EasyOCR and PaddleOCR

7.2 Optimized models

While the models have not been optimized yet, there will be a separate table to show its performance just as it has been done with the baseline models. A bar plot will also be drawn to display the differences between the Jetson Nano performance against the cluster GPUs performance.

7.3 Metrics computation

7.3.1 Strict Accuracy

As the name states, it is the most restrictive metric. It gives a binary outcome for each prediction, being 1 if the whole prediction string exactly matches the ground truth and 0 if it doesn't.

$$StrictAccuracy = \frac{1}{M}\sum_{i=1}^{M} 1(y_i = \hat{y}_i)$$

$$M = number\ of\ samples; y_i = ground\ truth\ for\ sample\ i;$$

$$\hat{y}_i = prediction\ for\ sample\ i; 1 = indicator\text{: } 1\ if\ true,\ 0\ else$$

7.3.2 Character Error Rate (CER)

This metric measures how different the predicted string is from the ground truth in terms of the minimum number of character insertions, deletions and substitutions that are needed to transform one into the other. It is the Levenshtein distance divided by the length of the ground truth and, as it, the bigger the value the more different the prediction is.

$$CER = \frac{S + D + I}{N}$$

$$S = number\ of\ substitutions; D = number\ of\ deletions;$$

$$I = number\ of\ insertions; N = number\ of\ characters\ in\ ground\ truth$$

7.3.3 Character-Level

These are computed as generic precision and recall metrics but treating each character as an item in a set. Precision computes how many of predicted characters were correct and recall computes how many of ground truth characters were correctly predicted.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

References

[1] EasyOCR. (n.d.). https://www.jaided.ai/easyocr/

[2] PaddleOCR. (n.d.). https://github.com/PaddlePaddle/PaddleOCR

[3] Google Cloud Vision API. (n.d.). https://cloud.google.com/vision/docs/ocr

[4] An overview of the Tesseract OCR engine. Smith, R. (2007). https://static.googleusercontent.com/media/research.google.com/es// *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, 629-633.

[5] Attention is All You Need. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). https://arxiv.org/pdf/1706.03762 *Advances in neural information processing systems*, *arXiv preprint arXiv:1706.03762*.

[6] Distilling task-specific knowledge from bert into simple neural networks. Tang, R., Lu, Y., Liu, L., Mou, L., Vechtomova, O., & Lin, J. (2019). https://arxiv.org/pdf/1903.12136 *arXiv preprint arXiv:1903.12136*.

[7] Q8bert: Quantized 8bit bert for natural language understanding. Zafrir, O., Boudoukh, G., Izsak, E., & Wasserblat, E. (2019). https://arxiv.org/pdf/1910.06188 *arXiv preprint arXiv:1910.06188*.

[8] On-device real-time hand gesture recognition with deep learning. Krekhov, A., & Alahi, A. (2020). https://arxiv.org/pdf/2111.00038 *arXiv preprint arXiv:2111.00038*.

[9] Technical City. (2019). GeForce RTX 2080 Ti vs Jetson Nano GPU.

[10] NVIDIA. (2024). JetPack SDK 5.1.

[11] TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models. Minghao Li and Tengchao Lv and Lei Cui and Yijuan Lu and Dinei Florencio and Cha Zhang and Zhoujun Li and Furu Wei. (2021). *arXiv preprint arXiv:2109.10282*. https://arxiv.org/abs/2109.10282

[12] PP-OCRv2: Ultra Lightweight OCR System. Du, Y., Jin, L., & Lin, T. (2021). *arXiv preprint arXiv:2109.03144*. https://arxiv.org/abs/2109.03144

[13] SVTRv2: Improved Scene Text Recognition via Visual Representation Learning. Dai, W., et al. (2024). *arXiv preprint arXiv:2411.15858*. https://arxiv.org/abs/2411.15858

[14] Real-time Auto License Plate Recognition with Jetson Nano. Winter2897. (n.d.). https://github.com/winter2897/Real-time-Auto-License-Plate-Recognition-with-Jetson-Nano

[15] Why AR2020-based cameras are perfect for OCR enablement in warehouse automation. e-con Systems. Sankar, G. (2024, April 30). https://www.e-consystems.com/blog/camera/why-ar2020-based-cameras-are-perfect-for-ocr-enablement-in-warehouse-automation/

[16] SmartCam T1023. SmartCow. (n.d.). SmartCam T1023 — Compact AI Camera with NVIDIA Jetson Nano. https://www.smartcow.ai/products/ai-cameras/smartcam-t1023

[17] NVIDIA. (2025). NVIDIA TensorRT SDK. https://developer.nvidia.com/tensorrt