Methodological Lessons in Deep Learning Pipeline Design: A Case Study of Cloud-to-Local Pivot in Sign-to-Voice Translation Systems

Vinod Kumar Kumaravel
MS in Artificial Intelligence
Yeshiva University
New York, USA
vkumarav@mail.yu.edu

Eesha Reddy Alluri
MS in Artificial Intelligence
Yeshiva University
New York, USA
ealluri@mail.yu.edu

Vishal Balaji
MS in Artificial Intelligence
Yeshiva University
New York, USA
vbalaji@mail.yu.edu

***Abstract*—This paper presents a critical analysis of deep learning pipeline design methodologies through the development of a Sign-to-Voice translation system. Initially, we proposed a cloud-native architecture leveraging Google Cloud Platform (GCP) services and large-scale datasets. However, this approach encountered fundamental limitations: extreme dataset scale (exceeding 1TB), heterogeneous data formats requiring complex preprocessing, and prohibitive computational requirements that impeded iterative experimentation. We document a methodological pivot to a local, model-centric pipeline that prioritizes controllability, modularity, and reproducibility. This transition enabled systematic exploration of three machine learning architectures—CNN, RNN/LSTM, and hybrid Transformers. Our final implementation achieves 98.7% classification accuracy and real-time inference capabilities, demonstrating that local pipelines can match or exceed cloud-based approaches while providing superior development transparency.**

I. Introduction

Sign-to-Voice translation systems convert American Sign Language (ASL) fingerspelling gestures into audible speech. This project's evolution from a cloud-native to a local, model-centric pipeline offers valuable methodological insights into deep learning system design.

A. Problem Domain and Original Proposal

Our initial proposal envisioned a cloud-native architecture leveraging GCP services—Cloud Storage for dataset management, Vertex AI for training, and Text-to-Speech API for voice synthesis—utilizing the Google ASL Fingerspelling Recognition Competition dataset.

B. Why the Cloud Approach Failed

The cloud-native approach encountered fundamental limitations:

- **Dataset Scale:** The 1TB+ dataset required 30-60 minute downloads per iteration and high storage costs.
- **Pipeline Opacity:** Distributed services made debugging difficult with limited visibility into intermediate states.
- **Computational Constraints:** Job launch overheads (5-10 minutes) discouraged rapid iteration.
- **Reproducibility:** Service versions and API changes evolved independently, complicating exact reproduction.

C. The Pivot: Local Model-Centric Pipeline

We pivoted to a local, model-centric pipeline prioritizing controllability and reproducibility. This choice optimized for research objectives, enabling rapid iteration (minutes vs. hours) and transparent debugging through version-controlled code and data.

II. Related Work

Cloud-native ML platforms offer managed services but introduce reproducibility challenges through service updates and cost structures that discourage extensive experimentation. Sign language recognition research, such as the Google ASL Competition, achieved state-of-the-art performance but requires substantial resources. Koller et al. demonstrated that lightweight CNN architectures could achieve competitive performance suitable for mobile deployment, which aligns with our final design philosophy.

III. Our Solution

A. Description of Dataset

1) Original vs. Final Dataset: The original Google ASL dataset (1TB+) was found incompatible due to its scale and complex 3D landmark preprocessing. We transitioned to the **Sign Language MNIST** dataset (100MB), which provides 28x28 grayscale images across 24 classes (A-Y excluding J, Z).

2) Custom Dataset Creation: We constructed a custom image dataset by capturing hand-sign images using team members' webcams. This dataset (1,500-2,000 images) was essential for validating model performance on real-world data and ensuring domain adaptation. It included varied lighting and backgrounds to bridge the gap between curated datasets and deployment.

B. *Preprocessing Pipeline*

We filter to 24 classes, reshape vectors to 28x28x1 tensors, and normalize pixels to [0, 1]. Data augmentation includes rotation, translation, and zoom. For real-world deployment, we implemented *Adaptive Gaussian Thresholding*, which improved live accuracy from 35% to 87.3% by emphasizing hand silhouettes.

C. *Machine Learning Algorithms*

We implemented three architectures: 1) **CNN:** Optimized for spatial pattern recognition with 476K parameters. 2) **RNN/LSTM:** Modeling temporal patterns by treating image rows as sequences. 3) **Transformer:** Utilizing Multi-Head Attention to capture global pixel relationships.

D. *Implementation Details and Migration*

The project was originally developed as a modular Python system. For final submission, it was migrated to a monolithic Jupyter Notebook (`.ipynb`). This migration required re-running the entire pipeline, including thresholding and training, within cloud-based kernels. Due to session timeouts, the reported metrics reflect the converged states achieved during the local development phase, subsequently validated through a successful live demonstration.

IV. Comparison

The CNN model emerged as optimal, providing the best accuracy-efficiency tradeoff.

TABLE I
MODEL PERFORMANCE COMPARISON

Model	Test Acc (%)	Params	Inference (ms)
CNN	98.7	476,264	0.83
RNN/LSTM	97.3	2,145,520	8.69
Transformer	97.1	3,452,352	40.62

TABLE II
CLOUD VS. LOCAL PIPELINE COMPARISON

Metric	Cloud	Local
Dev Velocity	5-10 min/iter	30 sec/iter
Debugging	Opaque	Transparent
Cost	$200+/month	One-time
Reproducibility	Variable	Complete

V. Future Directions

A substantial 6-12 month research extension would involve: 1) **Character-to-Sentence Transition:** Moving to continuous sign stream processing using sequence-to-sequence models and language modeling. 2) **Domain Adaptation:** Using adversarial learning to further bridge the dataset-to-real-world gap. 3) **Edge Optimization:** Quantization for deployment on mobile devices using TensorFlow Lite.

VI. Conclusion

Character-level recognition is well-solved with 98.7% test accuracy and 87.3% real-world accuracy. Our pivot from a cloud-native to a local pipeline demonstrated that superior development transparency and reproducibility can be achieved without compromising performance. The CNN model proved most suitable for real-time deployment due to its high efficiency and low latency.

References