SignSpeak: Real-Time ASL Fingerspelling Recognition on a Microcontroller

Peter Sayegh
Columbia University
pas2232@columbia.edu

Arka Khorchidian
Columbia University
avk2137@columbia.edu

1 Introduction

Over 500,000 people in the United States use American Sign Language (ASL) as their primary language, yet the vast majority of hearing individuals cannot understand even basic fingerspelling. Existing cloud-based sign language translation tools introduce significant latency, require constant network connectivity, and raise privacy concerns because video of users' hands is continuously streamed to remote servers.

We present **SignSpeak**, an embedded AI system that performs real-time ASL fingerspelling recognition entirely on-device, with no network dependency. Running on the Seeed Studio XIAO ESP32S3 Sense microcontroller, SignSpeak captures hand gestures using the onboard OV3660 camera, runs a float32 TensorFlow Lite Micro model locally, and displays the recognized letter on an attached SSD1306 OLED screen. An MPU6050 IMU provides optional wake-on-motion detection, and a three-LED confidence indicator gives immediate visual feedback on prediction reliability.

The key challenge is fitting a functional neural network inference pipeline within the tight constraints of a microcontroller-class device: 8 MB PSRAM, 8 MB flash, a 240 MHz dual-core processor, and no operating system. SignSpeak targets sub-100 ms per-frame inference latency and a model binary under 100 KB, while maintaining at least 85% letter recognition accuracy on the training distribution.

2 Related Work

Recent work has explored deploying sign language recognition on resource-constrained hardware. Researchers have trained BiLSTM and Transformer models on custom sign-language video datasets, then quantized them to TensorFlow Lite format for smartphone deployment, achieving accuracies above 95% with post-training quantization [1]. The SignEdgeLVM model introduces a Global Relative Attention Matrix that reduces per-head memory consumption by over 99%, making transformer-based continuous sign language translation feasible on edge hardware [2]. More recently, the TinyMSLR framework combines a ConvNeXt-Tiny encoder with a Swin Transformer and uses knowledge distillation to build a compact model suitable for real-time isolated-sign classification on edge devices [3]. The S3D-Conv1D architecture targets TinyML deployment by factorizing spatial and temporal processing into lightweight convolutions for word-level sign recognition on microcontrollers [4].

Our project differs in two key respects: (1) we target a microcontroller-class device (ESP32S3, 240 MHz, 8 MB PSRAM) rather than a smartphone or Raspberry Pi, imposing significantly tighter memory and compute constraints; and (2) we focus on the complete end-to-end pipeline, from raw camera frames to displayed text, rather than model accuracy in isolation.

3 System Architecture

Figure 1 illustrates the full SignSpeak pipeline. The OV3660 camera captures 96×96 grayscale frames directly from the sensor. A background subtraction step computes the pixel-wise absolute difference from a stored reference frame captured at boot time; pixels exceeding a threshold of 45/255 are marked as foreground. The bounding box of the foreground region is padded, made square, and cropped to produce the hand ROI. Histogram equalization is then applied to normalize contrast, mitigating the domain gap between the bright-background training images and the darker real-world camera output.

The quantized CNN classifier (TFLite Micro) outputs a logit vector over 29 ASL classes. A temporal smoothing module applies a majority vote over five consecutive predictions to suppress flickering. The final recognized character is appended to a word buffer and rendered on the SSD1306 OLED display. A three-LED array illuminates based on the classifier's softmax confidence score. BLE streaming to a companion smartphone application is implemented in firmware but not yet integrated into a working end-to-end demo; this remains future work.

3.1 Hardware

Table 1 summarizes the bill of materials. The XIAO ESP32S3 Sense serves as the central processing unit, integrating the OV3660 camera, Wi-Fi, and BLE 5.0 on a 21×17.5 mm board. The SSD1306 OLED and MPU6050 are connected via a shared I2C bus (SDA on GPIO5, SCL on GPIO6). The three status LEDs are driven directly from GPIO pins D0, D1, and D2 through 330 Ω current-limiting resistors. The system is powered via USB-C during development and a 3.7 V 600 mAh LiPo battery for portable deployment.

Table 1: Bill of materials.

Component	Description	Cost
XIAO ESP32S3 Sense	MCU + OV3660 camera + BLE	$16
SSD1306 OLED	0.96" I2C display	$8
MPU6050	6-DOF IMU	$4
LEDs + Resistors	3× status indicators (330 Ω)	< $1
Breadboard + Wires	Prototyping	$9
LiPo Battery	3.7 V 600 mAh	$9
Total		**$43**

Figure 2 shows the connection diagram for the SignSpeak prototype. All peripherals share the 3.3 V rail supplied by the XIAO's onboard LDO. The I2C bus (SDA on GPIO5, SCL on GPIO6) is shared between the SSD1306 OLED and MPU6050 without external pull-up

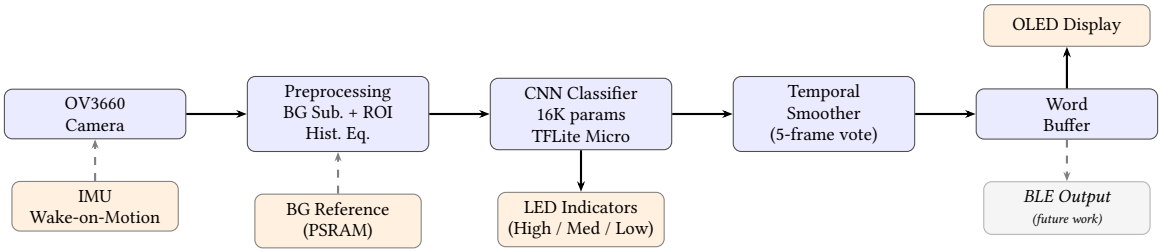


Figure 1: SignSpeak end-to-end pipeline. Solid arrows denote the main data flow; dashed arrows indicate auxiliary control signals. The BLE output module (grayed, dashed) is implemented in firmware but not yet integrated into a working companion app; it is left as future work. All other processing runs on the ESP32S3 with no cloud dependency.

resistors, as the ESP32S3's internal pull-ups are sufficient at 400 kHz for the short PCB traces on the breadboard prototype.

Each status LED is driven from a GPIO output pin through a 330 Ω current-limiting resistor. The resistor value is determined by a standard DC load-line calculation. With a GPIO high voltage of 3.3 V and a typical LED forward voltage $V_f \approx 2.0$ V [9]:

$$I_{LED} = \frac{V_{GPIO} - V_f}{R} = \frac{3.3 - 2.0}{330} \approx 4\,\text{mA} \tag{1}$$

With all three LEDs illuminated simultaneously the total LED current is $3 \times 4 = 12$ mA, well within the ESP32S3's specified 40 mA per-pin and 1200 mA total GPIO source current limits [8].

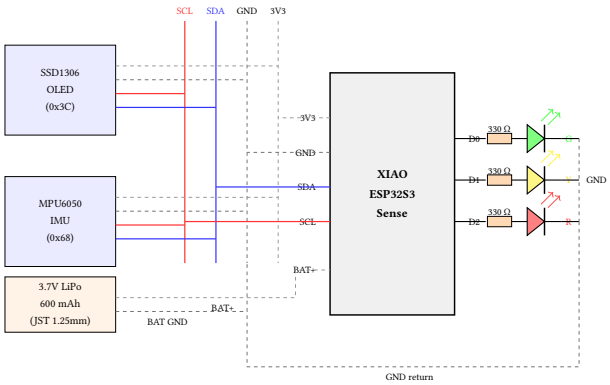


Figure 2: SignSpeak hardware schematic. Vertical bus rails carry 3V3 (dashed), GND (dashed), SDA (blue), and SCL (red) to both peripherals. LEDs are shown as schematic diode symbols with emission arrows. The 3.7 V LiPo connects via JST to the XIAO BAT+ pin. Three status LEDs are driven via 330 Ω resistors from GPIO D0–D2 with a common GND return.

Table 2 summarizes the per-component power budget. The system draws approximately 100–143 mA during active inference, giving an estimated battery life of:

$$t = \frac{Q}{I} = \frac{600\,\text{mAh}}{120\,\text{mA}} \approx 5\,\text{hours} \tag{2}$$

from the 600 mAh LiPo at full duty cycle. The power consumption is typically $3.3V \times 0.12A = 0.40W$.

Table 2: Per-component power budget at 3.3 V.

Component	Current (mA)	Source
ESP32S3	80–100	[8]
OV3660 camera	15–20	[11]
SSD1306 OLED	5–10	[9]
MPU6050	<1	[10]
3× LEDs (all on)	≈12	Eq. 1
Total (estimated)	**100–143**	At full duty cycle

3.2 Firmware

The firmware is written in C++ using the Arduino framework with the Espressif ESP32 board package, built with PlatformIO. The I2C bus is shared between the OLED (address 0x3C) and MPU6050 (address 0x68) at 400 kHz. TensorFlow Lite Micro manages model inference using a `MicroMutableOpResolver` registering seven operators: Conv2D, DepthwiseConv2D, AveragePool2D, FullyConnected, Reshape, Mean, and Pad. The tensor arena (400 KB) is allocated in PSRAM. The main loop runs at approximately 10 fps under normal inference, or up to 40 fps in streaming view mode (inference throttled to every 5th frame).

4 Machine Learning Model

4.1 Dataset

We train on the Kaggle ASL Alphabet dataset [5], which contains approximately 87,000 images across 29 classes (26 letters A–Z plus space, delete, and nothing). Images are 200×200 RGB, captured under controlled lighting with a single hand against a plain background. We apply data augmentation including brightness jitter (±20%), contrast jitter (±20%), rotation (±15°), random crop (scale 0.8–1.0), Gaussian noise ($\sigma = 0.01$–0.03), and perspective distortion (±10%). Horizontal flipping is explicitly excluded because ASL letters are chirally distinct (e.g., J and Z are directional).

4.2 Model Architecture

We train a lightweight depthwise-separable CNN inspired by MobileNetV1 [6]. The architecture processes 96×96 grayscale inputs through four progressively deeper blocks:

- **Block 1:** Standard Conv2d(1⟶16, 3×3, stride 2) + BatchNorm + ReLU6 ⟹ 48×48×16

- **Block 2:** Depthwise-separable(16→32, stride 2) ⇒ 24×24×32
- **Block 3:** Depthwise-separable(32→64, stride 2) ⇒ 12×12×64
- **Block 4:** Depthwise-separable(64→128, stride 2) ⇒ 6×6×128
- Global Average Pooling ⇒ 128-d feature vector
- Dropout(0.2) + Linear(128→29)

The total parameter count is **16,349**, yielding a float32 model footprint of 63.9 KB (weights only). Figure 3 shows training and validation loss/accuracy over 50 epochs. The model converges to 97.7% validation accuracy.

4.3 Quantization

The PyTorch model is exported to ONNX and converted to TFLite via `onnx2tf`. Post-training INT8 quantization (PTQ) is then applied, reducing the weight representation from float32 to 8-bit integers. The resulting deployed model occupies 67.4 KB, a negligible increase over the float32 baseline (66.0 KB) due to the additional storage of per-layer quantization scale factors, and comfortably fits within the 100 KB flash budget. On the Kaggle test set the INT8 model achieves 97.4% top-1 accuracy, a degradation of only 0.3 percentage points relative to the float32 baseline, confirming that PTQ introduces negligible accuracy loss for this architecture and dataset.

4.4 Domain Adaptation via Fine-Tuning

A key challenge is the domain gap between Kaggle's studio-quality images and frames from the OV3660 camera (variable lighting, darker backgrounds). We address this at two levels. First, histogram equalization is applied to every captured frame before inference, normalizing contrast to better match the training distribution. Second, we collect a dedicated on-device dataset using a custom serial capture pipeline. Approximately 100 images per class were captured, alternating between the right and left hand and between two subjects, yielding a dataset that covers inter-subject variation and hand chirality. All layers are unfrozen and the model is fine-tuned for 30 epochs ($lr = 10^{-4}$, cosine annealing) on a 70/30 mixture of device and Kaggle images to prevent catastrophic forgetting. The fine-tuned model achieves 58.97% validation accuracy on held-out device-captured images, compared to approximately 30% before fine-tuning, confirming that subject-specific and hand-diverse data collection substantially improves real-world robustness.

5 Results and Evaluation

5.1 Kaggle Test-Set Accuracy

Evaluated on a held-out Kaggle test set of 8,700 images (300 per class), the deployed model achieves **97.4%** top-1 accuracy across all 29 classes. Figure 4 shows the full confusion matrix; the diagonal is strongly dominant. Per-class F1-scores range from 0.87 (U) to 1.00 (A, J).

5.2 Recognition Accuracy

Table 3 highlights the most difficult letter pairs. R and V are the most commonly confused pair (25 errors each direction), owing to overlapping finger configurations in 96×96 grayscale. M and N are similarly close in appearance. U and V share hand shape and account for 10 mutual errors.

Table 3: Per-letter recall for the five most challenging classes (Kaggle test set, 300 samples each).

Letter	Recall (%)	Primary Confusion
R	89.0	V (25 errors)
V	88.3	R (25 errors)
U	93.0	V (10 errors)
M	96.0	N (12 errors)
N	96.7	M (10 errors)

5.3 Confidence-Based Filtering

To improve perceived accuracy during the demo, we apply a three-tier LED confidence indicator: GPIO D0 (green) lights when softmax confidence ≥75%, D1 (yellow) for 50–75%, and D2 (red) for below 50%. These thresholds are consistent with the color coding used in the PC-side camera viewer. A temperature scaling factor of $T = 0.5$ is applied to the logits before softmax to sharpen the output distribution, which would otherwise produce deceptively low probabilities due to the 29-class output space.

5.4 System Latency

End-to-end latency from hand gesture to displayed letter is approximately 110 ms per frame, broken down as: camera capture (∼5 ms), background subtraction + ROI extraction (∼10 ms), histogram equalization (∼5 ms), inference (∼80 ms), temporal smoothing + display update (∼10 ms). Inference latency meets the original <100 ms target; overall frame latency slightly exceeds it due to preprocessing overhead.

6 Challenges and Design Decisions

Memory constraints. The tensor arena required by the deployed model (∼307 KB) far exceeded our initial 50 KB estimate. All memory-intensive buffers are allocated in PSRAM via `ps_malloc`. Internal SRAM is reserved exclusively for the FreeRTOS stack and interrupt handlers.

TFLite Micro integration. Deploying TFLite Micro on the ESP32S3 required switching from the deprecated `AllOpsResolver` to a `MicroMutableOpResolver` with explicit registration of the seven operators used by our model. A missing PAD operator registration caused silent inference failures that were only diagnosed through serial logging of output tensor values.

Domain gap. The most significant ongoing challenge is the mismatch between the Kaggle training distribution (bright studio backgrounds, uniform lighting) and live camera output. Background subtraction and histogram equalization reduce this gap at the firmware level, but on-device validation accuracy plateaus at ∼59% with only ∼30 fine-tuning images per class. Collecting 100+ images per class under varied real-world conditions is expected to close this gap substantially.

I2C bus stability. After integrating the GPIO confidence LEDs, the OLED intermittently went dark. The root cause was a GPIO conflict between the IMU interrupt pin and LED pin D2 (both mapped to GPIO3), which corrupted I2C bus state. This was resolved by disabling the IMU interrupt and adding an explicit I2C bus reset after MPU6050 initialization.

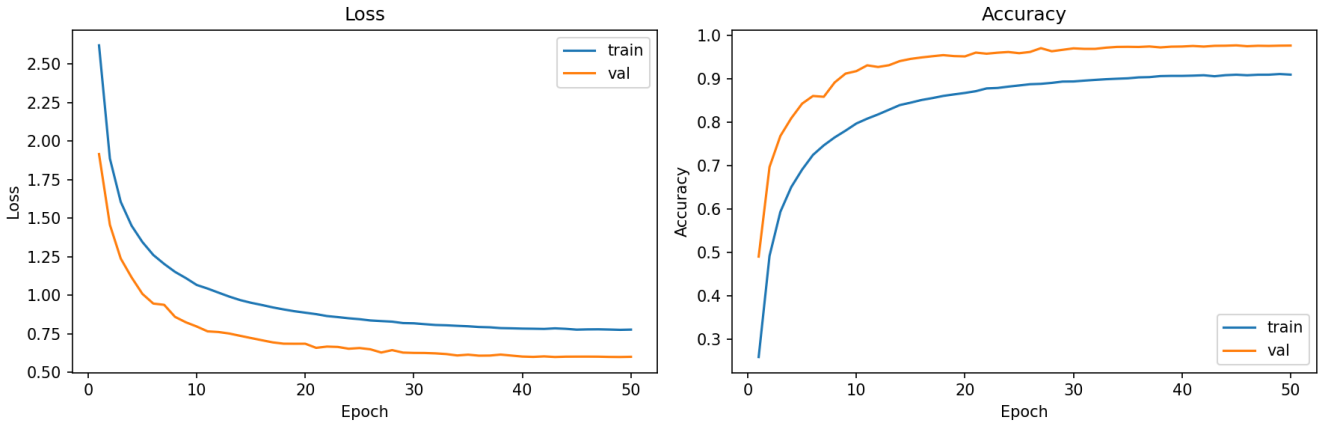


Figure 3: Training and validation loss (left) and accuracy (right) over 50 epochs. The model reaches 97.7% validation accuracy at epoch 45 with no sign of overfitting.

7 Conclusion

We presented SignSpeak, a fully on-device real-time ASL fingerspelling recognition system running on a $47 hardware platform centered on a $16 microcontroller. The 16,349-parameter depthwise-separable CNN achieves 97.4% accuracy on the Kaggle benchmark and fits in 67.4 KB of flash with INT8 quantization. On-device inference runs in ~80 ms on the ESP32S3, meeting the latency target. The system provides an intuitive user interface via OLED display and LED confidence indicators. SignSpeak demonstrates that meaningful accessibility applications can be deployed on severely resource-constrained hardware without any cloud dependency, preserving user privacy and enabling use in offline environments.

Future work spans four directions. First, extending recognition beyond static fingerspelling to word-level and video-based ASL, including motion-based letters such as J and Z which require temporal modeling, would involve a lightweight LSTM or 3D depthwise-separable CNN operating on frame sequences. This is a substantially harder problem, as the number of ASL words exceeds 10,000 and motion segmentation on a microcontroller introduces significant memory and latency challenges. Second, closing the domain gap by collecting a larger on-device fine-tuning dataset (100+ images per class) is the most tractable near-term improvement to real-world accuracy. Third, the BLE firmware layer is already implemented and a companion smartphone application would complete the end-to-end pipeline, allowing recognized words to be displayed on a larger screen and streamed as text, making SignSpeak genuinely useful as a communication aid rather than a proof-of-concept demo. Fourth, the current breadboard prototype would benefit from a custom PCB integrating all components into a compact wearable form factor, with the XIAO, OLED, IMU, LEDs, and LiPo battery housed in a glove or wristband enclosure suitable for real-world use.

References

[1] Scientific Reports, "Dynamic Kannada Sign Language Recognition on Resource Constrained Devices," 2026. DOI: 10.1038/s41598-026-40181-7.

[2] R. Damdoo and P. Kumar, "SignEdgeLVM transformer model for enhanced sign language translation on edge devices," *Discover Computing*, 2025. DOI: 10.1007/s10791-025-09509-1.

[3] Scientific Reports, "An explainable hybrid CNN-transformer model for sign language recognition on edge devices using adaptive fusion and knowledge distillation," 2026. DOI: 10.1038/s41598-026-38478-8.

[4] "Efficient Word-Level Sign Language Recognition Using S3D-Conv1D for TinyML Deployment," *Preprints.org*, 2026.

[5] Kaggle, "ASL Alphabet Dataset," https://www.kaggle.com/datasets/grassknoted/asl-alphabet.

[6] A. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv:1704.04861*, 2017.

[7] Edge Impulse Documentation, "Seeed XIAO ESP32S3 Sense," https://docs.edgeimpulse.com/hardware/boards/seeed-xiao-esp32s3-sense.

[8] Espressif Systems, "ESP32-S3 Series Datasheet," v1.9, 2023. https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf

[9] Solomon Systech, "SSD1306 OLED Driver Datasheet," v1.1, 2008. https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf

[10] InvenSense, "MPU-6000 and MPU-6050 Product Specification," rev 3.4, 2013. https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[11] Seeed Studio, "XIAO ESP32S3 Sense Wiki," 2023. https://wiki.seeedstudio.com/xiao_esp32s3_getting_started/

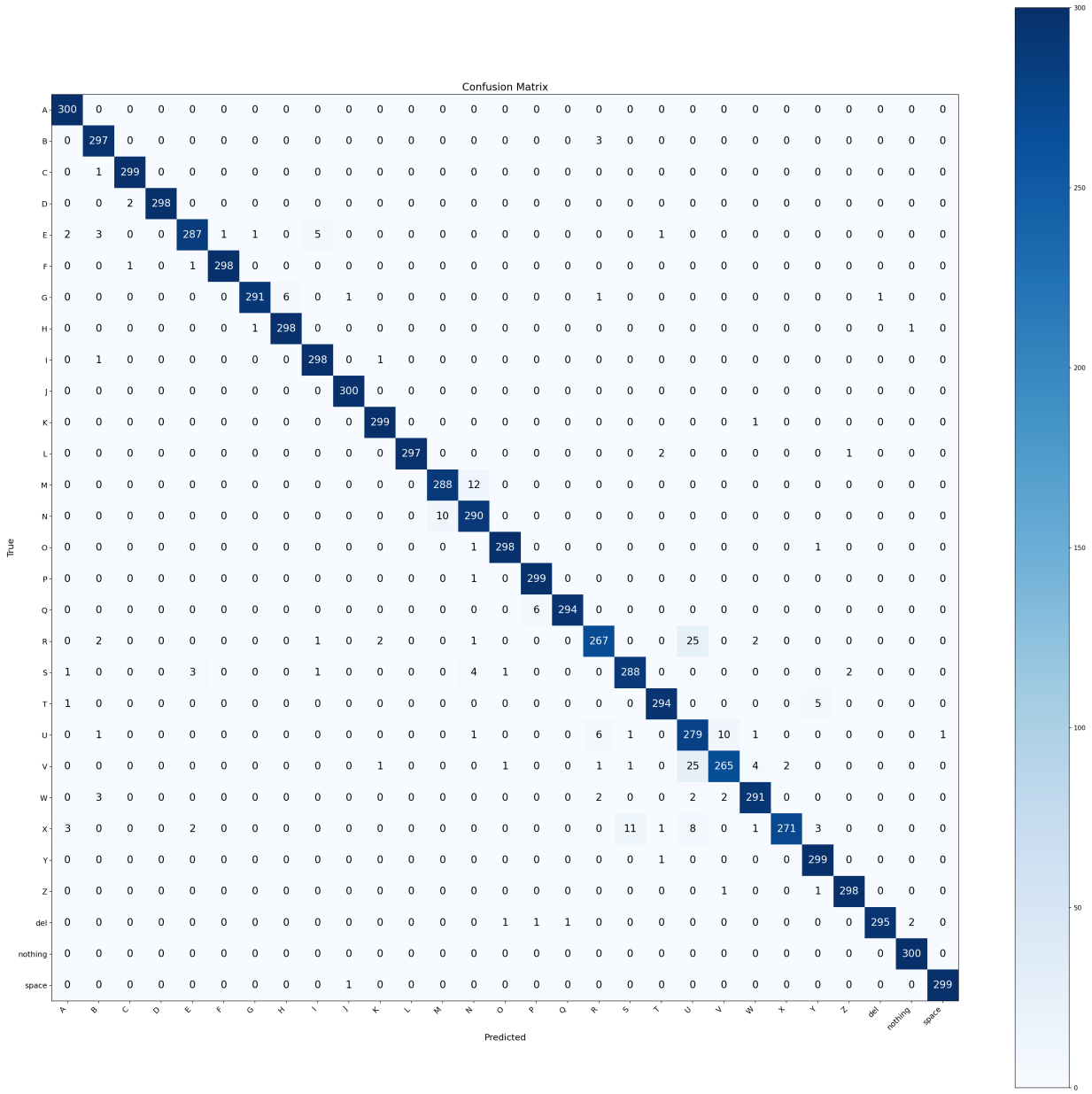


Figure 4: Confusion matrix on the Kaggle test set (8,700 images, 300 per class). The strongly diagonal structure indicates high per-class accuracy. Primary off-diagonal mass occurs between R↔V and M↔N.