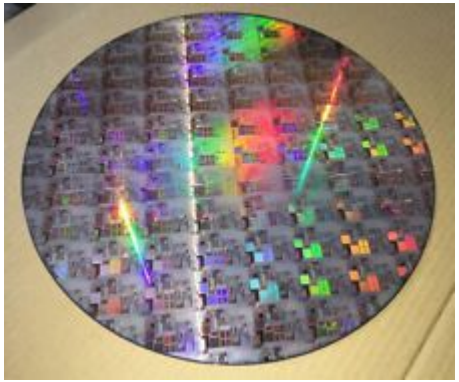
Multimodal Anomaly Detection in Silicon Wafer Manufacturing Using Image and Sensor Data

Jack Peng, Robert Alvarez, Jayden Cheung, Arsh Batth

Introduction

Introduction

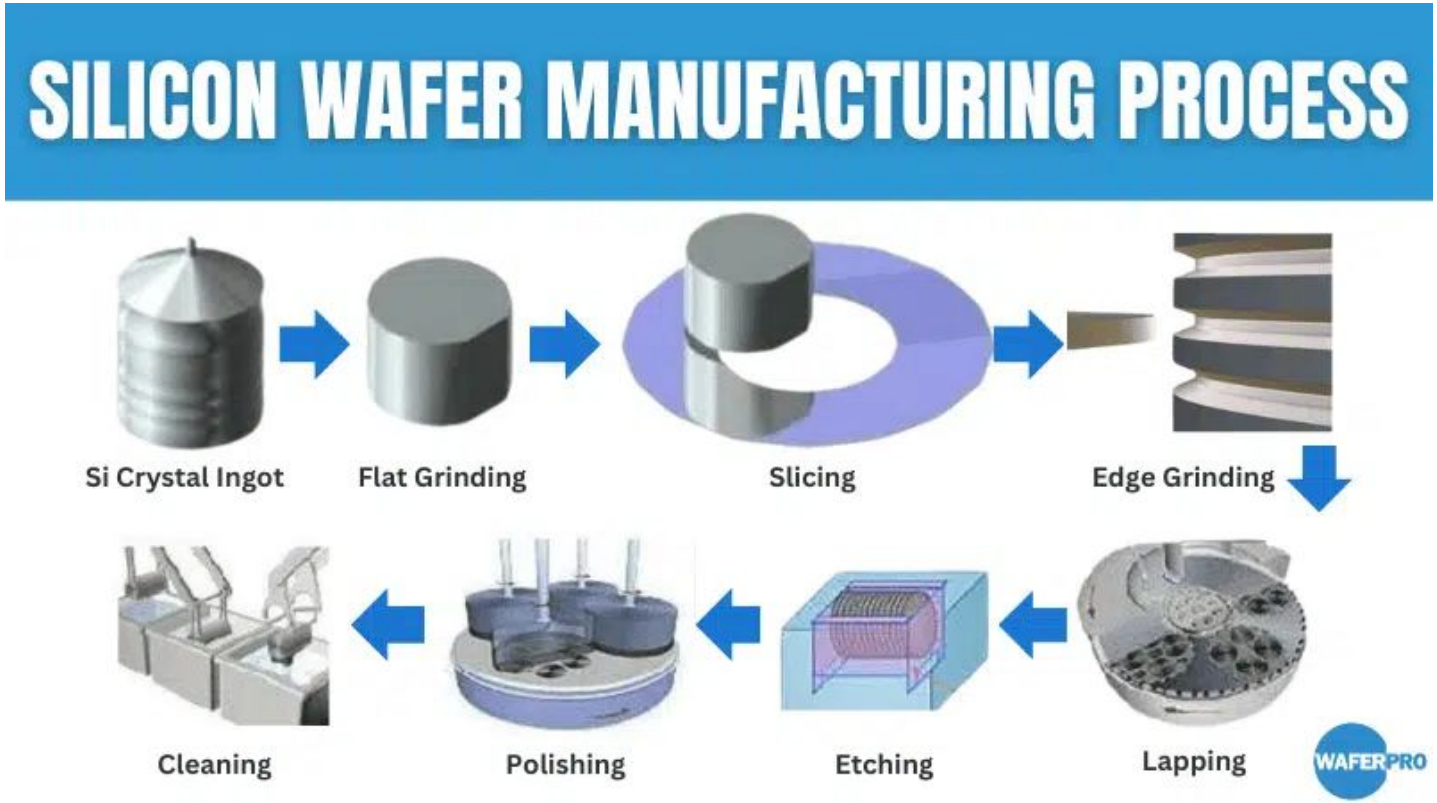
- What is a wafer?
  - thin, circular slice of semiconductor (silicon) which is used to create electronic components
  - base upon which microchips and other electronic devices are built
  - especially relevant to hard drive construction



Introduction

- Wafer manufacturing is a highly complex process that involves numerous steps and precise control over environmental conditions. Defects in silicon wafers can result in significant losses in yield and quality. Traditional inspection methods rely on visual assessment or sensor-based anomaly detection, but each has limitations.
- The manufacturing process can take months for a single batch, meaning that defects can propagate for months before being detected
    - it is crucial in this process that defective batches are detected and discarded early on to avoid wasted effort further in the pipeline
- This study is an exploration of the possibility of a multi-modal approach that integrates image analysis and sensor data for a more robust anomaly detection system

Introduction



Presentation Outline

- Literature Survey
  - An Ensembled Anomaly Detector for Wafer Fault Detection
  - Review of Wafer Surface Defect Detection Methods
  - Anomaly Detection and Segmentation for Wafer Defect Patterns Using Deep Convolutional Encoder–Decoder Neural Network Architectures in Semiconductor Manufacturing.
- Our Research Findings

Literature Survey: Related Works

Related work 1

Giuseppe Furnari, Francesco Vattiato, Dario Allegra, Filippo Luigi Maria Milotta, Alessandro Orofino, Rosetta Rizzo, Rosaria Angela De Palo, and Filippo Stanco. 2021. An Ensembled Anomaly Detector for Wafer Fault Detection. Sensors 21, 16 (2021), 5465. https://doi.org/10.3390/s21165465

Wafer Fault Detection in Semiconductor Manufacturing

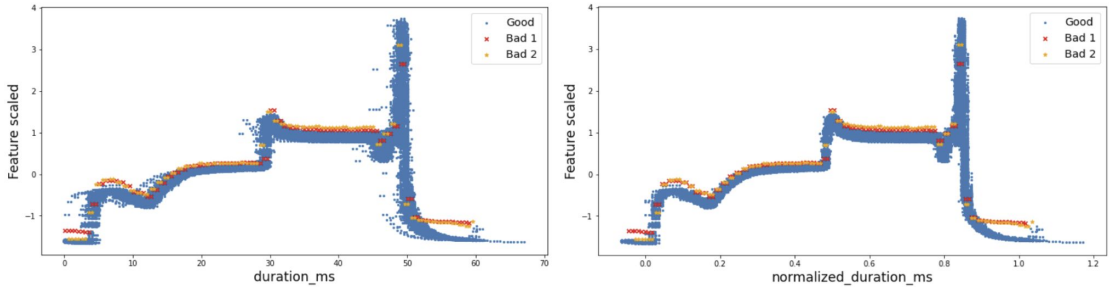
1. **Complex Process:** Multiple phases (diffusion, parametric tests, assembly, final test).
2. **Anomaly Detection Critical:** Yield losses often due to subtle parameter interactions.
3. **Traditional SPC Limitations:** Fail to detect faults from multivariate interactions.

Proposed Ensembled Anomaly Detector:

1. **Hybrid Approach:** Combined Univariate (ANOVA) and Multivariate (OCSVM) analyses.
2. **Voting System:** Balances insights from multiple analytical methods.
3. **Three Balancing Strategies Evaluated:** Equally Weighted (EWC), Statistic-based (MSC), Score-based (SBC).

Datasets & Preprocessing

1. **Two Real-world Datasets:** Dataset-1 (~1 million samples, 44 parameters), Dataset-2 (~125k samples, 22 parameters)
2. **Preprocessing Steps:** Standardization (z-score), time normalization (aligns time series data), removal of meaningless measurements.
3. **Time Normalization Impact:** Enhances Separation of anomalies.

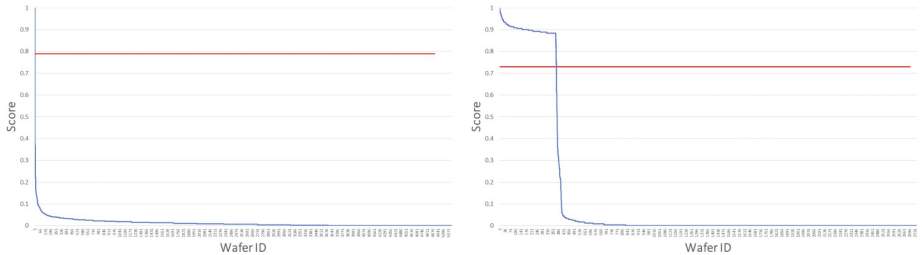


(**a**) Raw timeseries for a sample feature. (**b**) Normalized time series for a sample feature.

Figure 1. Time preprocessing: (**a**) the original raw time series and (**b**) the normalized representation, given a sample feature. Notice how, thanks to time preprocessing, the good samples are clustered together in a more compact baseline, while outliers are more separated from the baseline.

Results - Evaluating Balancing Strategies

1. **SBC Outperformed:** Clearly distinguished normal vs abnormal wafers/
2. **SBC Reduced False Alarms:** D1 (0 false alarms), D2 (only 5 false alarms)
3. **Confusion Matrix Highlights:** SBC had highest accuracy compared to EWC & MSC.



(e) Score-Based Criterion applied to D1 (f) Score-Based Criterion applied to D2

Table 1. Confusion matrix for all balancing criteria and both datasets. For each confusion matrix we have true negatives (true abnormal Wafer-ID detected) and false negative (normal wafer classified as abnormal) on the first row, while on the second row we have the false positive (abnormal wafer classified as normal) and true positive (true normal wafer).

		D1		D2	
	Actual / Predicted	Abnormal	Normal	Abnormal	Normal
EWC	Abnormal	2	1	367	37
	Normal	0	5101	0	752
MSC	Abnormal	1	0	367	14
	Normal	1	5102	0	775
SBC	Abnormal	2	0	367	5
	Normal	0	5102	0	784

Comparison with Traditional Methods

1. **Benchmarking:** Compared against OCSVM, COPOD, FABOD, Isolation Forest (IF).
2. **Top Performance:** The ensemble approach matched or exceeded benchmarks.
3. **Best Anomaly Detection:** Detected all true anomalies in D1, minimal false alarms in D2.

Table 2. Comparison for Datasets D1 and D2 of our proposed method (PROPOSED) with several classic anomaly detection methods: OCSVM, COPOD, FABOD, and IF. The comparison is presented as a benchmark of confusion matrices, reporting True Negative (TN), False Positive (FP), True Positive (TP), and False Negative (FN).

DATASET	METHOD	TN	FP	TP	FN
D1	PROPOSED	2	0	5102	0
	OCSVM [21]	0	2	5102	0
	COPOD [25]	0	2	5102	0
	FABOD [16]	2	0	5102	0
	IF [26]	0	2	5102	0
D2	PROPOSED	367	0	784	5
	OCSVM [21]	367	0	788	1
	COPOD [25]	189	178	788	1
	FABOD [16]	367	0	777	12
	IF [26]	367	0	731	58

Conclusions and Impact

1. **Innovative Hybrid Detection:** Effectively detects complex wafer anomalies.
2. **Practical Industrial Application:** Balances detailed fault detection with manageable false alarm rate.
3. **Future Directions:** Deep-learning extensions, advanced root-cause analytics.

Related work 2

Jianhong Ma, Tao Zhang, Cong Yang, Yangjie Cao, Lipeng Xie, Hui Tian, and

Xuexiang Li. 2023. Review of Wafer Surface Defect Detection Methods. Electronics

12, 8 (2023). https://doi.org/10.3390/electronics12081787

Review of Wafer Surface Defect Detection Methods

Wafer surface defect detection plays an important role in controlling product quality in semiconductor manufacturing.

This paper analyzes the research progress in the field of wafer surface defect detection, introducing the classification of wafer surface defect patterns and their causes, and dividing current mainstream methods into three categories:

1) methods based on image signal processing,
2) methods based on machine learning, and
3) methods based on deep learning.

https://doi.org/10.3390/electronics12081787

Introduction to Wafer Manufacturing

Silicon Wafer Production

Used in the manufacturing of semiconductor chips and other processes.

Defect Formation

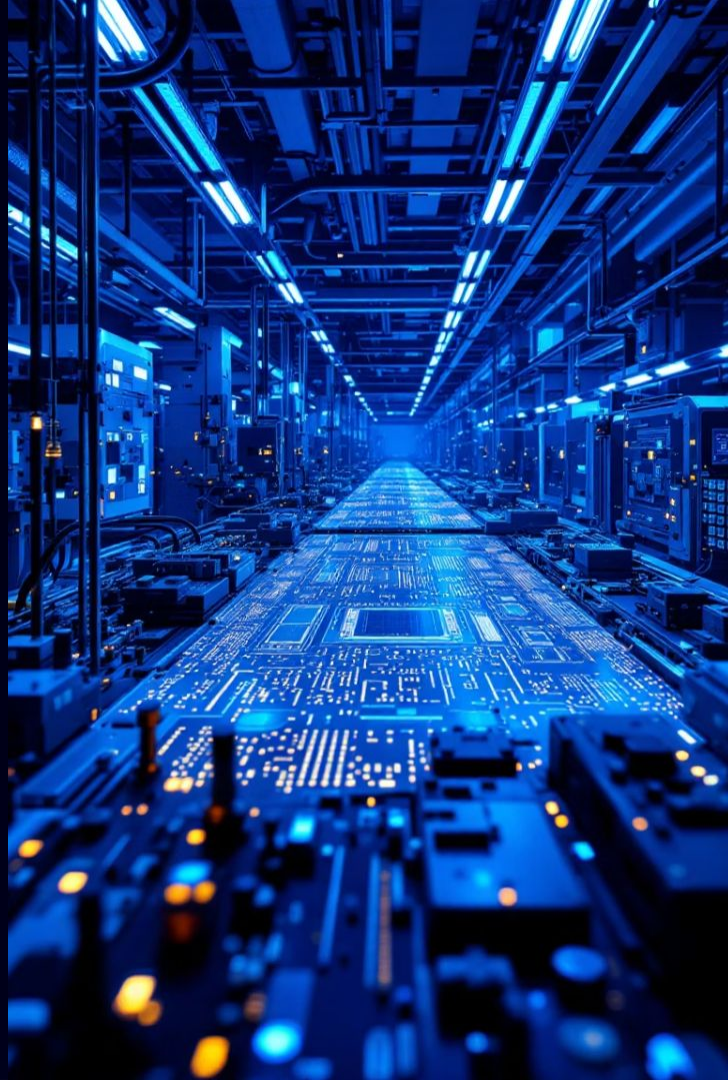
Environmental factors can generate defects on the wafer surface, affecting production yield.

Detection Methods

Early manual inspection has been replaced by computer vision methods.

Neural Networks

Neural networks have rapidly developed with high efficiency, accuracy, low cost, and strong objectivity in wafer defect detection.



Wafer Surface Defect Patterns

Unpatterned Wafer Defects

These mostly occur in the pre-lithography stage of wafer production and are caused by machine failures.

- Scratch defects
- Particle contamination

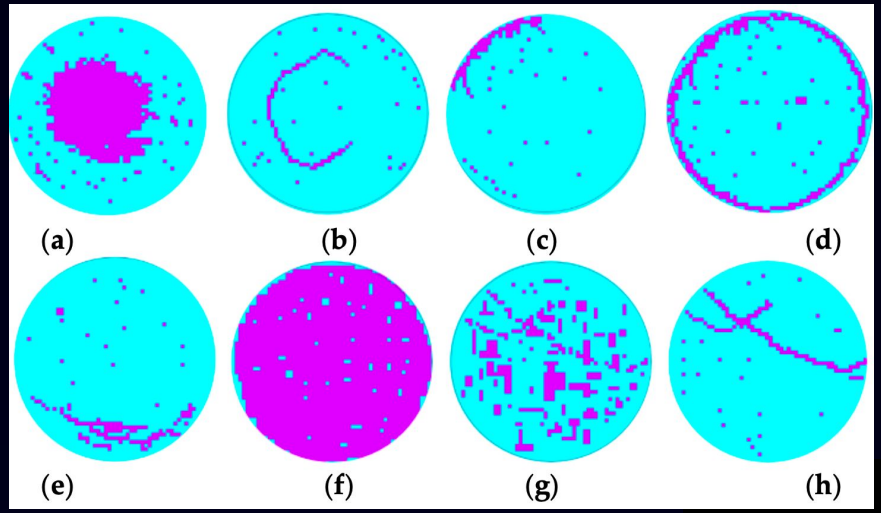
Patterned Wafer Defects

These are mostly found in the middle process of wafer production, caused by improper exposure time, development time, and post-baking time.

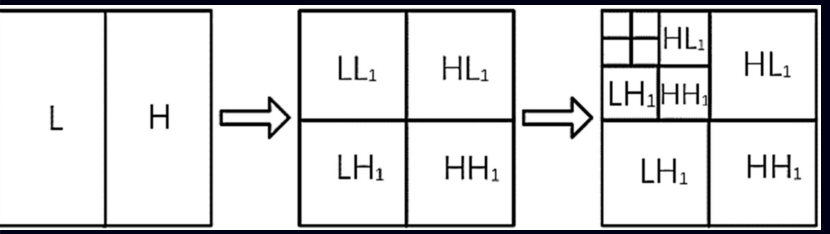
- Open circuit defects
- Short circuit defects
- Line contamination
- Bite defects

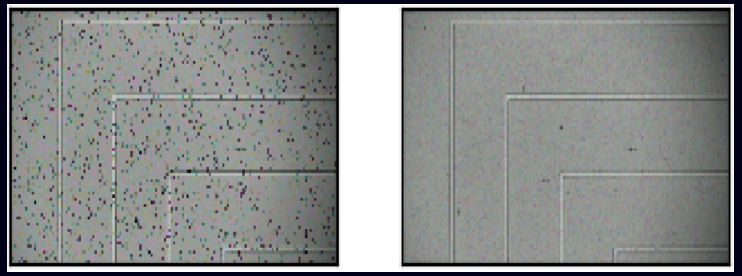
Types of Wafer Map Defect Patterns

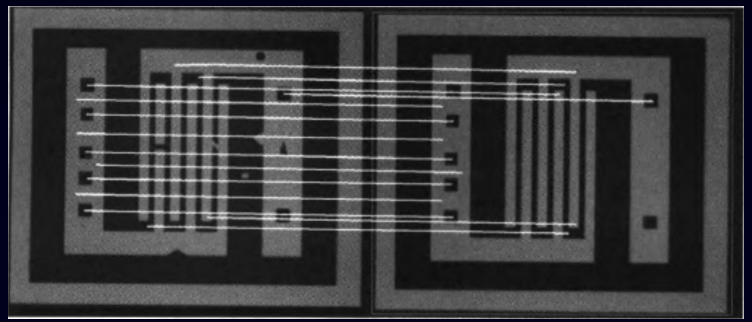
(a) Center,
(b) Donut,
(c) Edge-loc,
(d) Edge-ring,
(e) Local,
(f) Near-full,
(g) Random, and
(h) Scratch.



Wafer Surface Defect Detection Based on Image Signal Processing





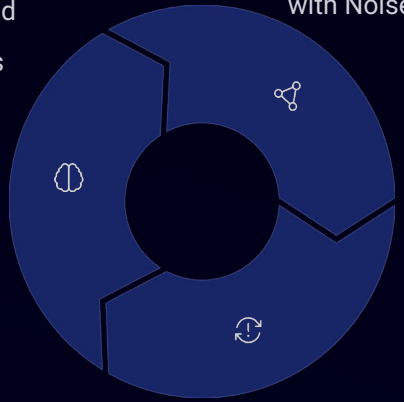


Model Algorithm	Innovation	Limitation
Wavelet transform	The image can be decomposed into multiple resolutions and presented as local sub-images with different spatial frequencies. Anti-grain.	The selection of the threshold is very dependent and the adaptability is poor.
Spatial filtering	Based on spatial convolution, remove high-frequency noise, and perform edge enhancement.	Performance depends on the threshold parameter.
Template matching	The template matching algorithm has strong anti-noise ability and fast calculation speed.	Sensitive to feature object size.

Machine Learning for Wafer Defect Detection

Supervised Learning

Model Algorithm	Innovation	Limitation
KNN	Insensitive to abnormal data and highly accurate.	High complexity and computation intensity.
Decision Tree-Radon	Apply Radon to form new defect features.	Overfitting is highly proficient.
SVM	SVM efficiently classifies multivariate, multimodal, and indivisible data points.	It is not friendly to multiple samples, and the kernel function is difficult to locate.

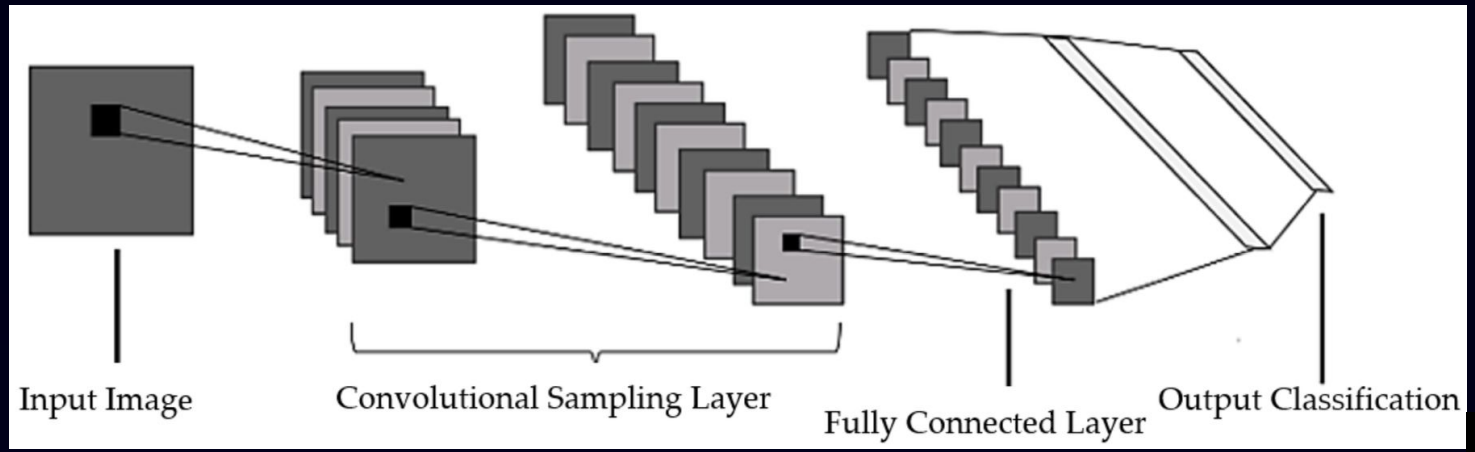


Unsupervised Learning

Model Algorithm	Innovation	Limitation
MLP-Clustering Algorithm	The MLP is used to enhance the feature extraction capability.	Depends on the choice of activation function.
Density-Based Spatial Clustering of Applications with Noise	Outliers can be selectively removed based on defect pattern characteristics.	The sample density is not uniform or the sample is too large, the convergence time is long, and the clustering effect is poor.

Deep Learning for Wafer Defect Detection

Model Algorithm	Innovation	Accuracy
DC-Net	The sampling area is focused on the defect feature area, which is very robust to mixed defects.	97.8%
CNN-Based Combined Classifier	Separately design classifiers for each defect, strong adaptability to new defect modes.	99.0%
Classification Retrieval Method Based on CNN	Simulated datasets can be generated to account for data imbalances.	98.9%



Challenges and Future Directions

Limited Public Datasets

Few high-quality public datasets exist due to high production and labeling costs.

Continuous Defect Evolution

New defects are continuously generated during fabrication.

Complex Hybrid Defects

Smaller feature sizes result in multiple defect types in a wafer, with defects folded together.

Universal Detection Models

Designing a universal network model to detect all defects across different wafer types would avoid resource waste from creating separate training models for each defect dataset.

Real-time Detection

Most defect detection models are offline and unable to meet real-time industrial production requirements.

Related work 3

Takeshi Nakazawa and Deepak V. Kulkarni. 2019. Anomaly Detection and Segmentation for Wafer Defect Patterns Using Deep Convolutional Encoder–Decoder Neural Network Architectures in Semiconductor Manufacturing. IEEE Transactions on Semiconductor Manufacturing 32, 2 (2019), 250–256. https://doi.org/10.1109/TSM.2019.2897690

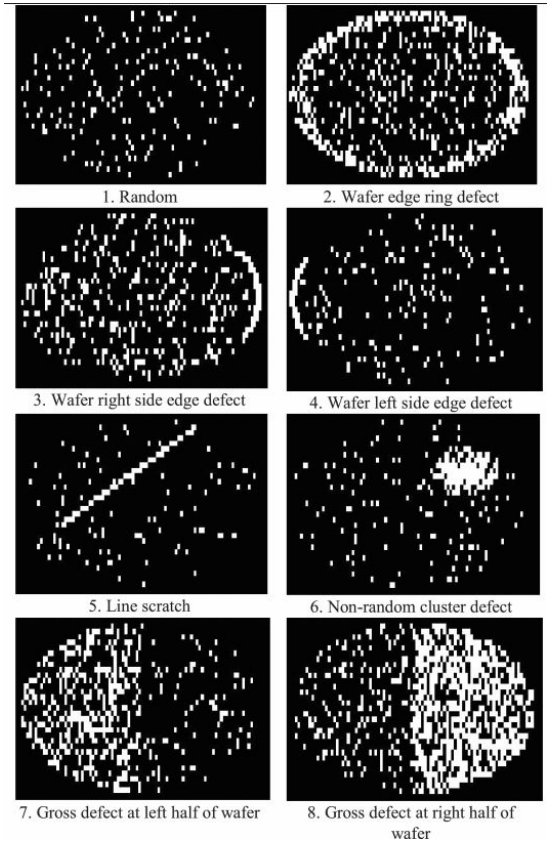
Detecting Wafer Defect Patterns with Deep CNNs

Motivation:

- Small defect clusters can scrap entire wafer batches
- Engineers use wafer maps for
  - Common-pattern analysis (root-cause)
  - Early anomaly detection (excursion prevention)
- Automated vision needed for speed, consistency, and novelty detection

Class label	Wafer map defect class name
C1	Random defect
C2	Wafer edge ring defect
C3	Wafer right side edge defect
C4	Wafer left side edge defect
C5	Line scratch defect
C6	Non-random cluster defect
C7	Gross defect at left half of wafer
C8	Gross defect at right half of wafer

The examples of the synthetic wafer maps used for model training and validation.

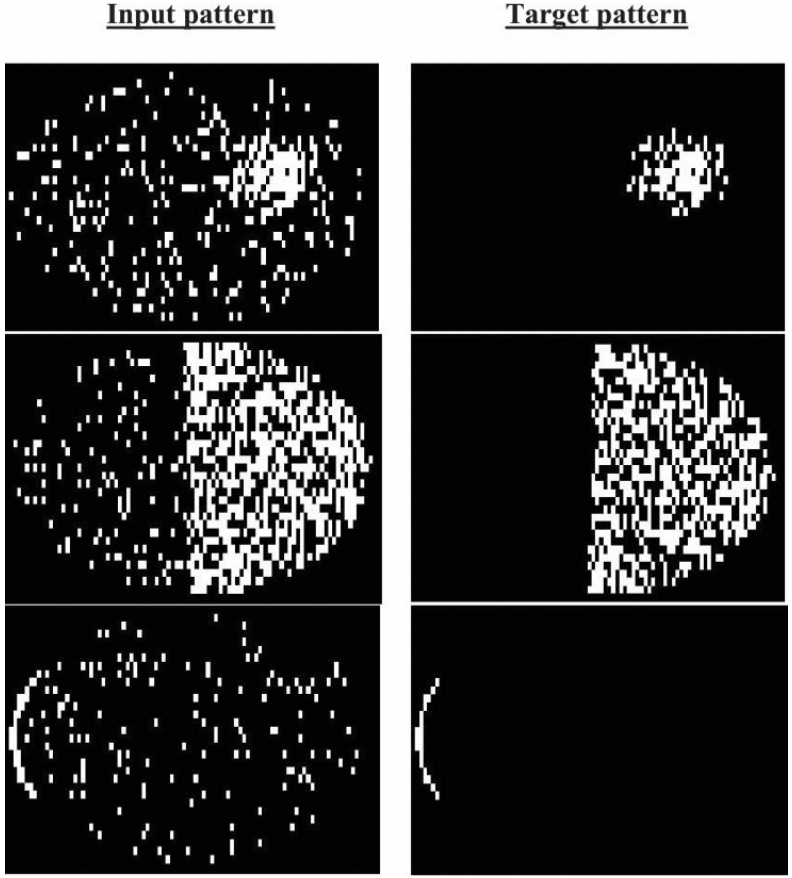


Building Synthetic Wafer Maps

- Defects simulated via Poisson point process

$$P(k, \Lambda) = \frac{\Lambda^k}{k!} e^{-\Lambda}$$

- **Background noise:** Random “blemish” defects scattered via a Poisson process.
- **Single target cluster:** Overlay exactly one of eight prototypical defect shapes.
- **Why synthetic?** Real anomalies are too rare and too varied to gather at scale.

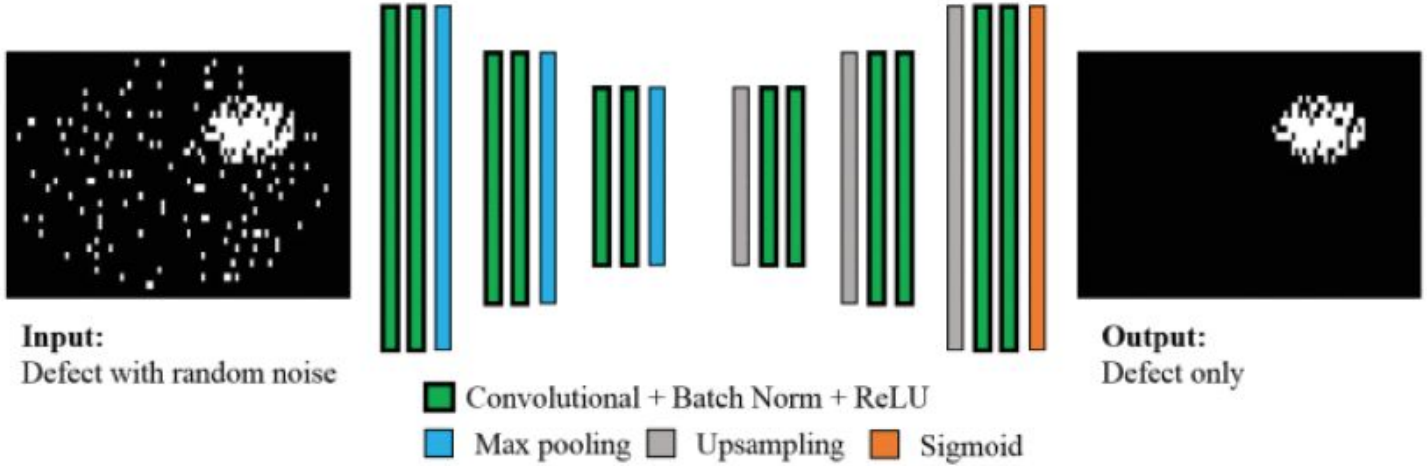


Input and target wafer map pattern pairs.

Model Architectures

SegNet: Learns to up-sample using saved “pooling” locations for crisp boundaries.

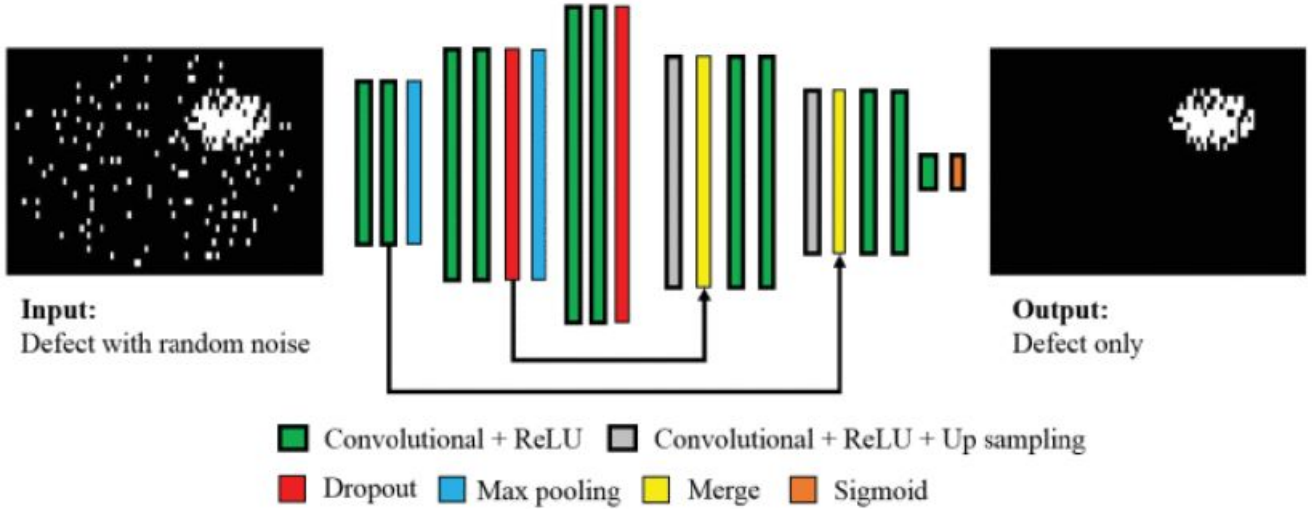
- Encoder: Two 3×3 conv → BatchNorm → ReLU → 2×2 max-pool (Channel sizes: 128→64→32)
- Decoder: 2×2 up-sample using saved indices → Two conv layers
- Output: Sigmoid & binary cross-entropy loss



Model Architectures

U-Net: Merges shallow (fine) and deep (coarse) features with skip-connections.

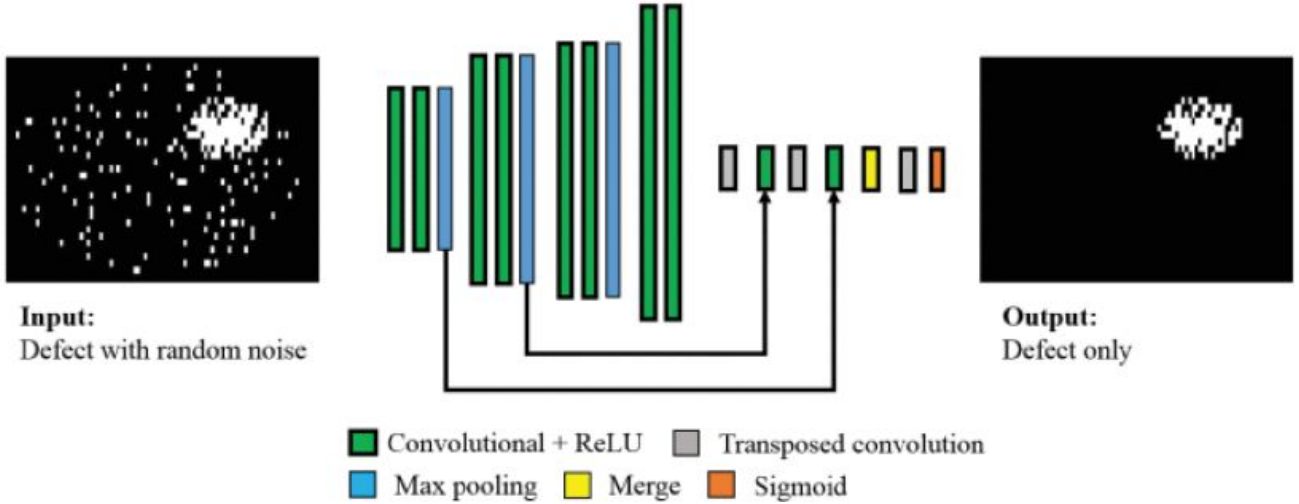
- Same conv blocks (32→64→128 channels)
- Skip-connections: Copies feature maps from encoder to decoder to preserve fine edges
- Ideal for small or thin defect clusters



Model Architectures

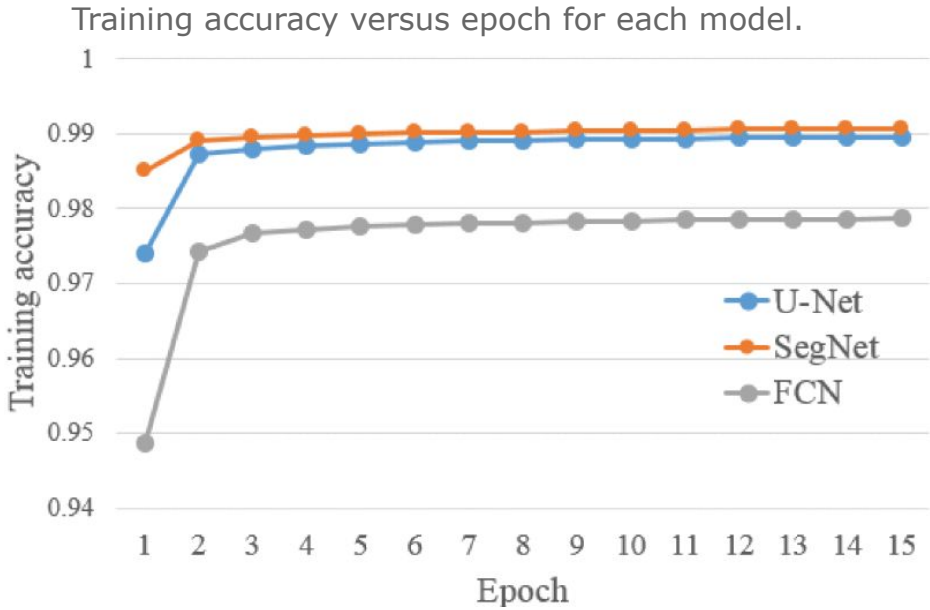
FCN: Uses learned transpose convolutions to grow feature maps back to full size.

- Two conv layers (32, 64, 128 channels) with stride 2
- Two conv layers (256 channels) with 7×7 and 1×1 filters
- Upsamples via learned transpose convolutions + merges from pooling layers



Training Efficiency

- 15 epochs, binary cross-entropy loss
- Converges by epoch 5:
  - SegNet 99.0%
  - U-Net 98.9%
  - FCN 97.8% train accuracy



Architecture	SegNet	U-Net	FCN
Avg. time/epoch [min]	19.9	10.0	4.2

the average training time in minutes per epoch.

Synthetic and Unseen Results

- IoU (Intersection over Union/Jaccard Index):

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

- mIoU: mean IoU across test samples
- mIoU > 0.5 = reliable segmentation
- Synthetic base tests (3 300 images):
  - SegNet/U-Net mIoU = 0.65–0.88 across 8 patterns
  - FCN mIoU = 0.33–0.80 (struggles on thin scratches)
- Unseen pattern tests (new shapes):
  - SegNet mIoU = 0.74–0.91
  - U-Net mIoU = 0.77–0.89
  - FCN mIoU = 0.61–0.77
- As defects cover more of the wafer, SegNet holds up better than U-Net in testing.

mIoU Result for Basis Pattern Test Set

Pattern\Architecture	SegNet	U-Net	FCN
Wafer edge ring	0.655+/-0.12	0.728+/-0.08	0.428+/-0.08
Right side edge	0.879+/-0.07	0.850+/-0.11	0.804+/-0.09
Left side edge	0.883+/-0.11	0.794+/-0.19	0.330+/-0.26
Line scratch	0.649+/-0.25	0.663+/-0.23	0.538+/-0.23
Non-random cluster	0.815+/-0.09	0.817+/-0.09	0.646+/-0.11
Gross defect at left	0.753+/-0.10	0.799+/-0.06	0.459+/-0.08
Gross defect at right	0.876+/-0.04	0.877+/-0.05	0.640+/-0.04

mIoU Result for Unseen Defect Patterns

Pattern\Architecture	SegNet	U-Net	FCN
Gross edge damage	0.912+/-0.05	0.894+/-0.07	0.776+/-0.08
Curved scratch	0.824+/-0.10	0.808+/-0.12	0.769+/-0.10
Line scratch with non-random cluster	0.743+/-0.08	0.771+/-0.09	0.610+/-0.09

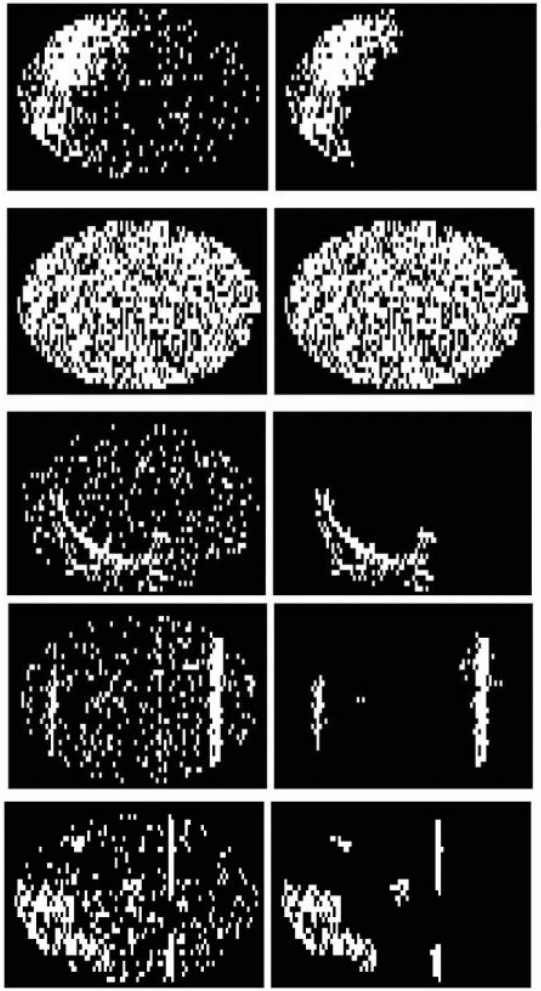
Real-Wafer Validation

Validation set: 1,191 real production wafers (22.9% with real clusters)

Results:

- 100% of true defect clusters detected
- 94.3% of purely noisy wafers correctly marked “no cluster”

Key takeaway: Synthetic-only training enables robust, in-line anomaly detection



The defect pattern examples from real wafer maps. The patterns from the real wafers (left) and inference results (right).

Scalability & Practical Insights

- Scaling synthetic data from 17 k → 26 k
- Training time per epoch: 19 min → 30 min
- mIoU improves: 0.796 → 0.828 (SegNet)
- Trade-off: modest time for measurable accuracy gain (a few hours to achieve reasonable performance)

Architecture	17,000 dataset	26,000 dataset
Avg. time/epoch [min]	19.9	30.5
Overall mIoU	0.796	0.828

Training Time in Minutes and mIoU Comparison

Conclusion

- Synthetic-only training works: trained on 17k generated maps, the model detects real clusters with 100% accuracy and ignores noise 94% of the time.
- Eliminates the need for thousands of rare, hand-labeled real-fault wafers.
- This lets fabs spot rare or brand-new problems before they cause big yield losses, without waiting for real failures.
- After detecting a defect, simple image tools can measure its location, size, and shape for root-cause debugging.
- The same idea—training on synthetic examples—could speed up other quality-inspection tasks (e.g., circuit boards, medical scans).

Our Findings

Outline

- Discussion of Model Fusion
- Image Based Detection
  - CNN
- Sensor Based Anomaly Detection
  - Single Class SVM
- Conclusion

Discussion of Model Fusion

- A fundamental obstacle to creating a multimodal anomaly detection system is a lack of synchronized data
  - No way to correlate the images and sensor data
  - while there are many wafer image datasets and many sensor datasets, we could not find any publicly available datasets which include both correlated sensor and image data
- Therefore our initial proposal remains feasible in theory, but difficult to implement in the absence of synchronized data
- Ultimately, we have created an evaluation of various modes of defect detection

Image Based Defect Detection - Convolutional Neural Network

Data set: WM-811k form kaggle.

kagglehub.dataset_download("qingyi/wm811k-wafer-map").

Input:

Image data store in numpy 2D arrays.

Image shapes:

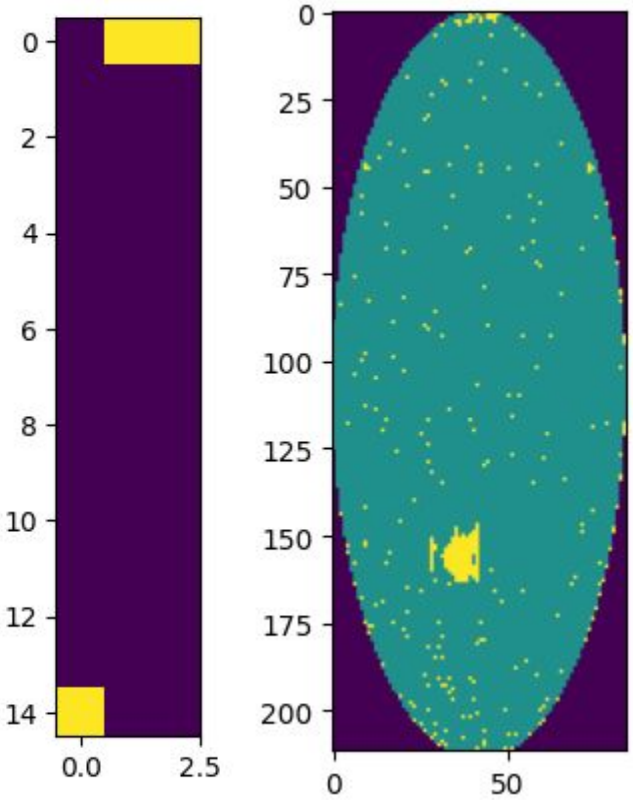
From (15, 3) to (212,84).

Avg: (35.23, 34.82).

Labels (failure types):

1. ‘Center’, 2. ‘Donut’, 3. ‘Edge-Loc’, 4. ‘Edge-Ring’,

5. ‘Loc’, 6. ‘Near-full’, 7. ‘Random’, 8. ‘Scratch’, 9. ‘none’



‘Loc’

Data Cleaning

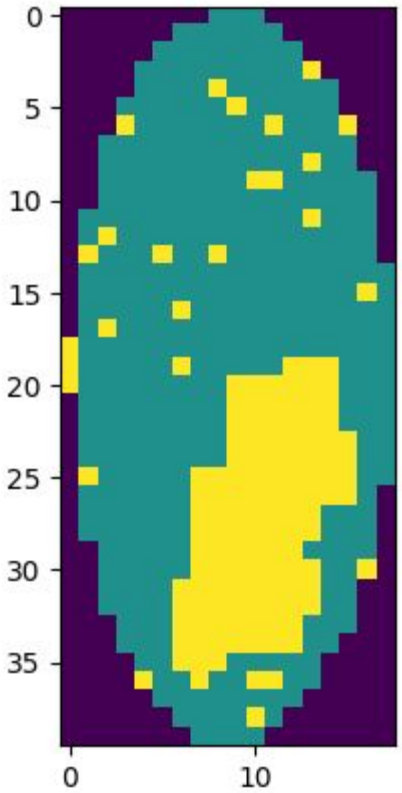
1. Remove samples with no failure label.

   Original number of samples: 811,457.

   Number of label samples: 172,950.

2. Remove images with a shape entry of less than 18.

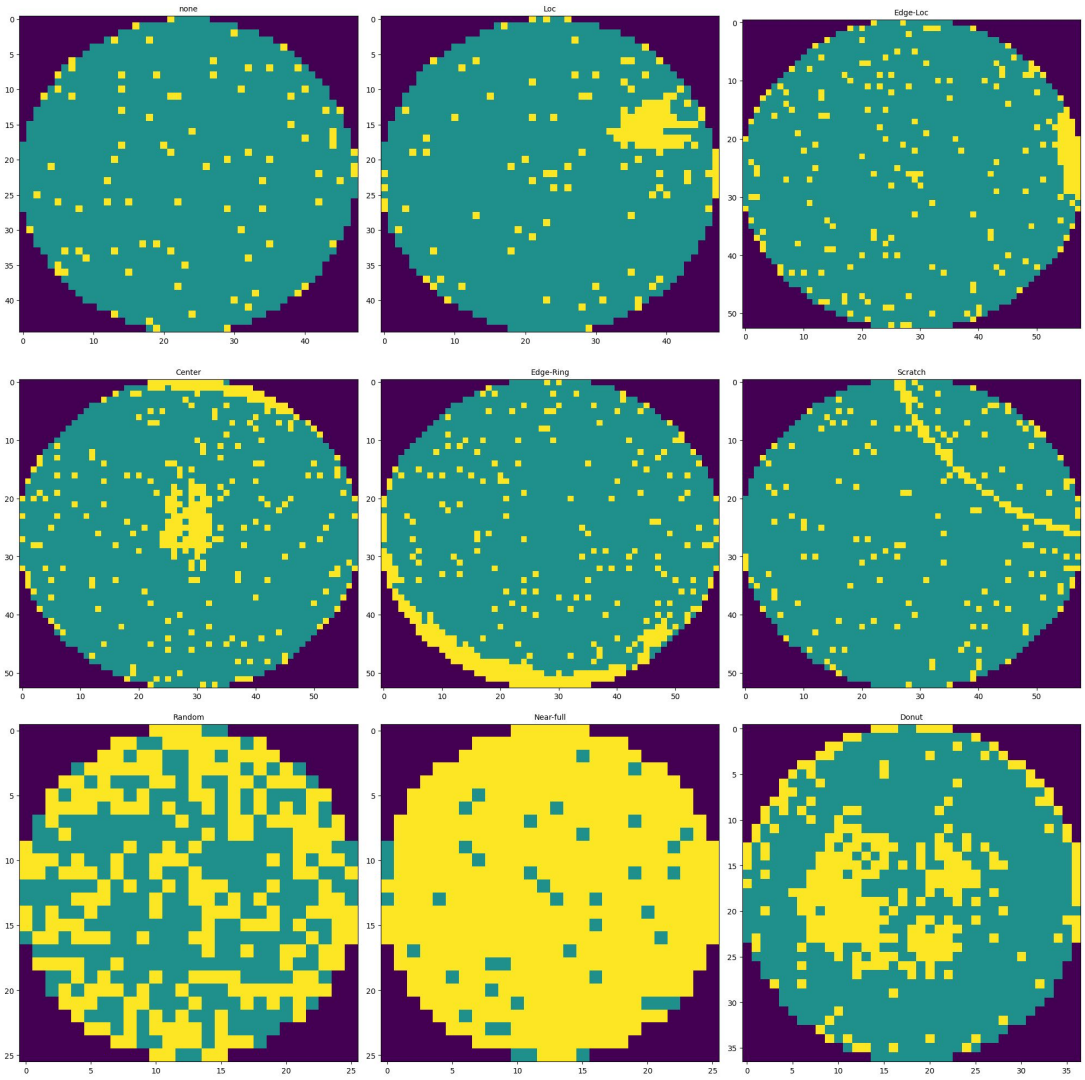
   Min shape: (15, 3) -> (40 18).



‘Loc’

Failure type random samples

None	Loc	Edge-Loc
Center	Edge-Ring	Scratch
Random	Near-Full	Donut



Data characteristics

Data characteristics:

- Each image consist of 1 channel.
- The data type is np.uint8.
- Possible values per pixel entry: [0, 1, 2].
- Shapes range from (40 18) to (212,84).
- Unbalanced data set:
  - Non-defective: 147,431 (85%)
  - Defective: 25,519 (15%)

Dataset incompatibility with CCNs + unbalanced solutions

Convolutional Neural Network limitation w/ this dataset:

- Regular CNN expects all images to be of the same shape.

Solutions:

1.
    a. CNN: Resize images using by convert it into PNGs using the PIL library.
    b. Data unbalance: Use 5,000 samples of defective and 5,000 of non-defective.
2.
    a. CNN: Use KNN to decide what color the new pixels have to be when resizing.
    b. Data unbalance: Create synthetic data from defective samples.

Approach 1: Resize images by convert it into PNGs.

Advantages:

- Image quality is enhance:
  - Number of channels from 1 to 3.
  - New range of values from [1,2,3] to [0, …, 255].
- Easy to set up.

Disadvantages:

- Slow to write images into hard disk because of its sizes.
  - New image shape are (128, 128).
  - Triple the number of channels (more memory).
  - Each entry takes now 32 bytes instead of 8 (more memory).
  - Data conversion for 10,000 images took a few hours in a CPU.

Convolutional Neural Network

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dense_1 (Dense)	(None, 1)	65

Architecture:

- # of trainable parameters: 101,569.

Training:

- Epochs: 10.
- Batch size: 32.
- Optimizer: “Adam”
  - A Method for Stochastic Optimization.
- Loss function: Binary cross entropy.
- Evaluation metric: Accuracy.
- Input size: 8k images.
- Time: Approximately 10 mins.

Evaluation

Precision: 0.938

Recall: 0.745

F1-Score: 0.830

AUC: 0.935

Test loss: 0.357

Test accuracy: 84.78%

Classification Report:

	precision	recall	f1-score	support
0	0.79	0.95	0.86	5000
1	0.94	0.75	0.83	5000
accuracy			0.85	10000
macro avg	0.86	0.85	0.85	10000
weighted avg	0.86	0.85	0.85	10000

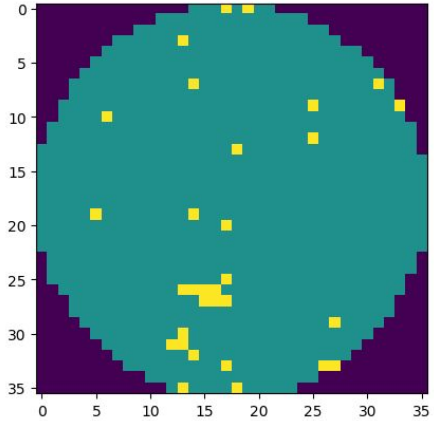
Approach 2: Use KNN to resize image.

Advantages:

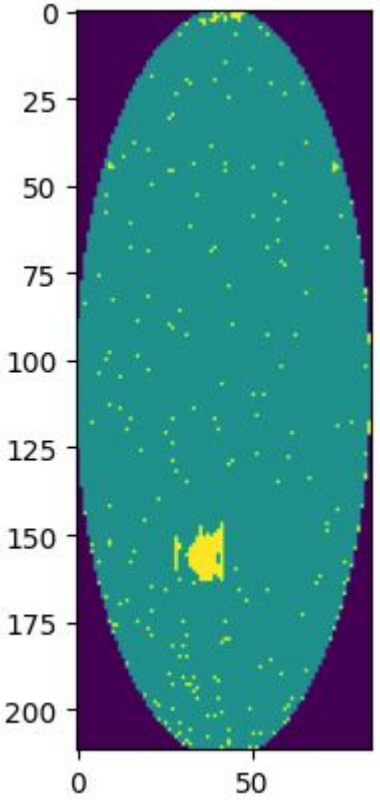
- Number of channels stays the same (1).
- Each entry still only takes 8 bytes.

Disadvantages:

- A knn approach doesn’t respect defects shape.



‘Loc’



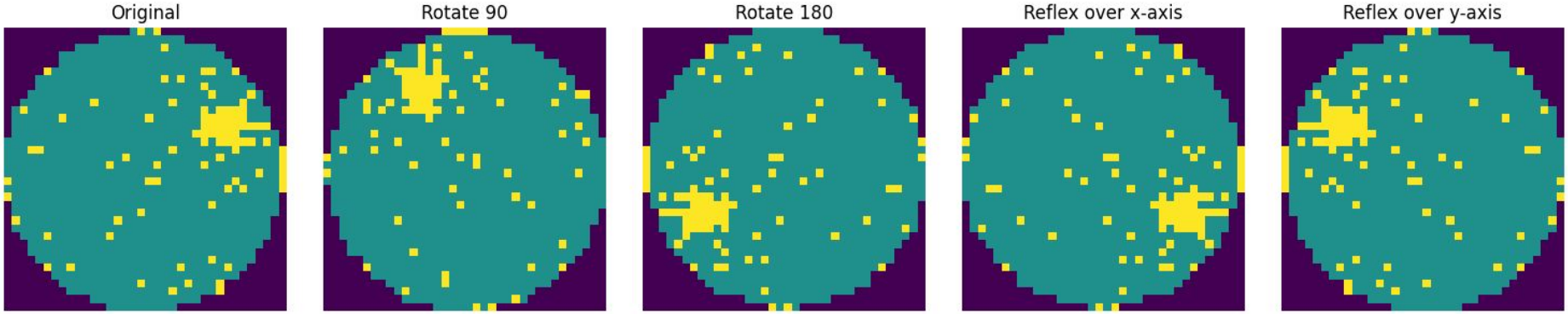
Synthetic data generation for defective cases

Possible transformation:

- Rotate image 90 degrees.
- Rotate image 180 degrees.
- Reflect image over the x-axis.
- Reflect image over the y-axis.

For each image:

- Decide at random between n=1 or n=2.
- Sample n random function without replacement.
- Each sample function was applied to the image.



Convolutional Neural Network

Architecture:

- # of trainable parameters: 103,218.

Training:

- Epochs: 15.
- Batch size: 64.
- Optimizer: SGD(lr=1e-3, p=0.9)
- Loss function: Cross entropy loss.
- Evaluation metric: Accuracy.
- Time: Approximately 8 mins.

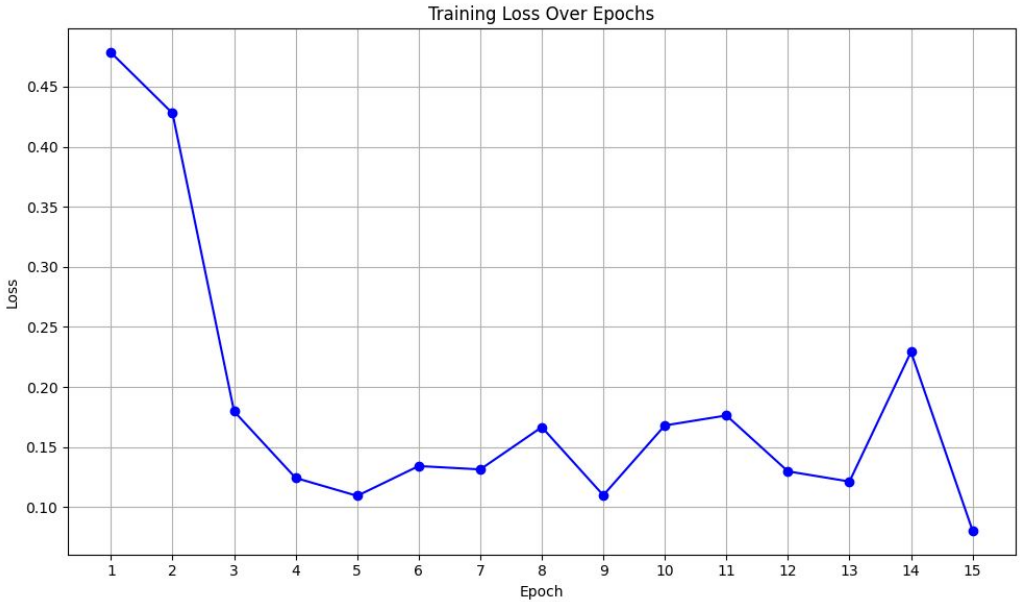
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 34, 34]	160
MaxPool2d-2	[-1, 16, 17, 17]	0
Conv2d-3	[-1, 16, 15, 15]	2,320
MaxPool2d-4	[-1, 16, 7, 7]	0
Linear-5	[-1, 128]	100,480
Linear-6	[-1, 2]	258

Evaluation

Train Loss: 0.063256

Train Acc: 97.89%

Test Acc: 96.95%



Sensor Data Based Detection - Single Class SVM

- What is One-Class SVM Doing?
  - “Here’s what normal looks like — now draw a tight boundary around it. Later, anything outside this boundary is probably defective."

Sensor Data Based Detection - Single Class SVM

- Data
  - taken from UCI SECOM dataset
  - SECOM = ***SE***mi***CO***nductor ***M***anufacturing
  - Preprocessing:
    - replace missing values with column averages
    - scale the data to have zero mean and unit variance

Sensor Data Based Detection - Single Class SVM

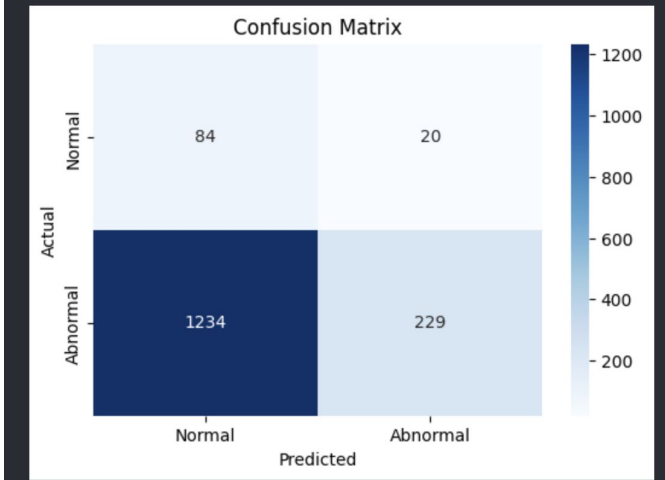
- Single class svm built using sklearn
- Has 2 parameters: gamma and nu
  - The SVM builds a “fence” around the normal points using a kernel funciton.
  - gamma controls how flexible the fence is.
  - nu is roughly how many “errors” we expect — it defines the tradeoff between a tight vs. loose boundary.
  - perform k-fold cross validation to find best values of gamma and nu

Sensor Data Based Detection - Single Class SVM

- Initial results
  - overall accuracy ~20%
  - predicts defective wafers as normal 94% of time
- Why?
  - Actually, accidentally flipped -1 and 1, which swapped defective and nondefective wafers

Classification Report:

	precision	recall	f1-score	support
-1	0.92	0.16	0.27	1463
1	0.06	0.81	0.12	104
accuracy			0.20	1567
macro avg	0.49	0.48	0.19	1567
weighted avg	0.86	0.20	0.26	1567



Sensor Data Based Detection - Single Class SVM

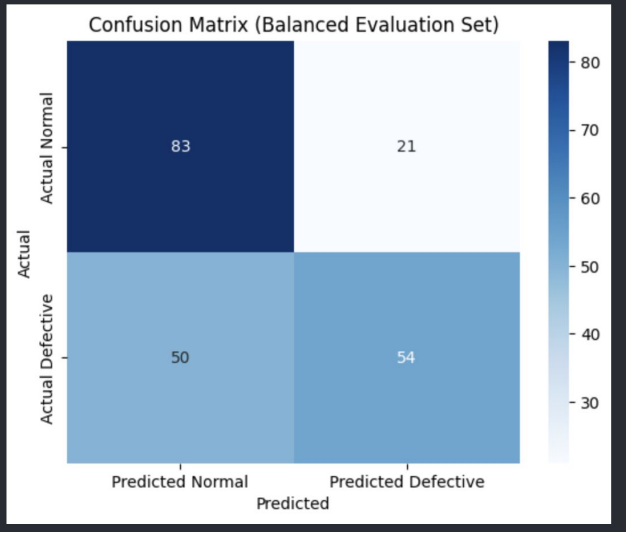
- Solution:
  - flip labels

```
df["label"] = df["label"] * -1
```

  -
  - also, undersample from nondefective data to have equal amounts of defective and nondefective data
- Results:
  - 0.7 f1-score
  - improved precision and recall for both defective and nondefective classes
  - results still not great
    - Why?
    - In high-dimensional spaces, data tends to become sparse (“curse of dimensionality”).
    - One-Class SVM relies on support vectors and distances — which become less meaningful in sparse spaces.
    - Result: it can overfit to noise or miss subtle defect patterns.

Classification Report (Balanced Evaluation Set):

	precision	recall	f1-score	support
-1	0.72	0.52	0.60	104
1	0.62	0.80	0.70	104
accuracy			0.66	208
macro avg	0.67	0.66	0.65	208
weighted avg	0.67	0.66	0.65	208



Conclusion

- While we were unable to train a multimodal defect detector, we were able to individually train models on sensor and image data
- The CNN model proved to be the most effective, being 96% accurate at classifying defective/nondefective wafers
- Sensor data was slightly more difficult to work with, due to the SECOM dataset being significantly skewed
    - Single Class SVM yielded about 66% accuracy

NLP TRANSFER LEARNING FOR LOW-DATA LANGUAGES

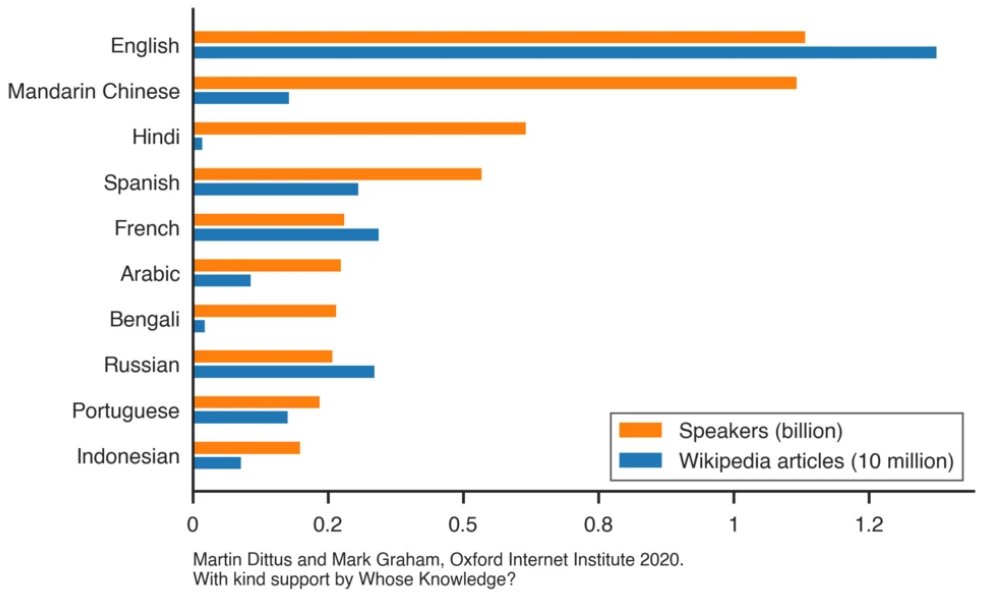
Joshua Kamphuis, Cristiano Battistini, Ruixin Wang, Xiaochen Yang

PURDUE UNIVERSITY | Department of Computer Science

Introduction

- Natural language data on the internet is massively skewed toward a few high-resource languages
- With significantly less data available for low-resource languages, accurate natural language processing models are difficult to create.

Language	Proportion of Internet
English	49.2%
Spanish	6.0%
German	5.7%
Japanese	5.1%
French	4.4%

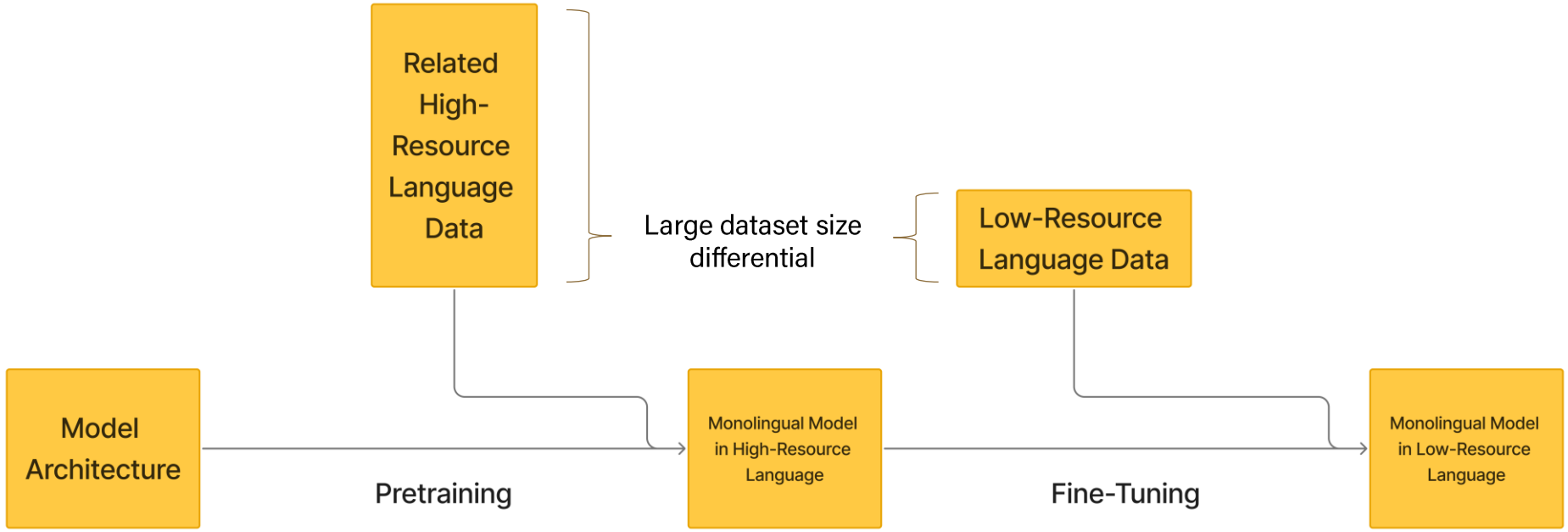


Above: Volume of Wikipedia articles in common languages (https://internetlanguages.org/en/numbers/wikipedia-language-geography/#gallery-1)

Left: Top 6 languages used in websites across the internet (https://w3techs.com/technologies/overview/content_language).

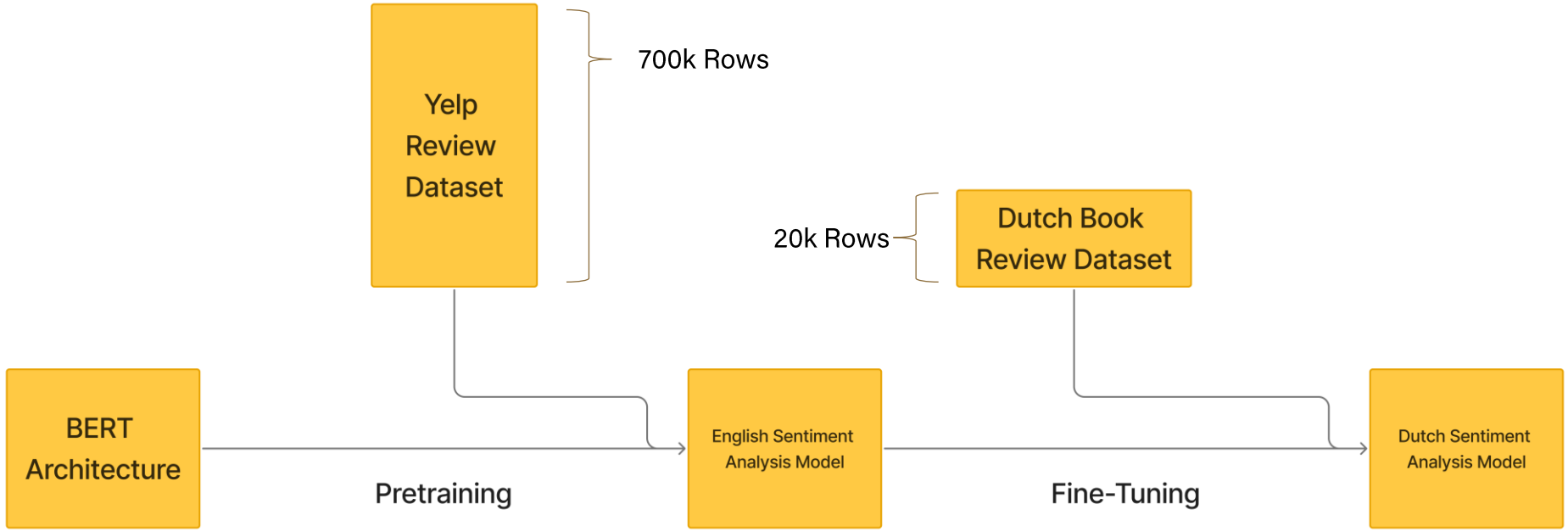
Our Proposal

Can high-resource language data be leveraged for training models in low-resource languages?



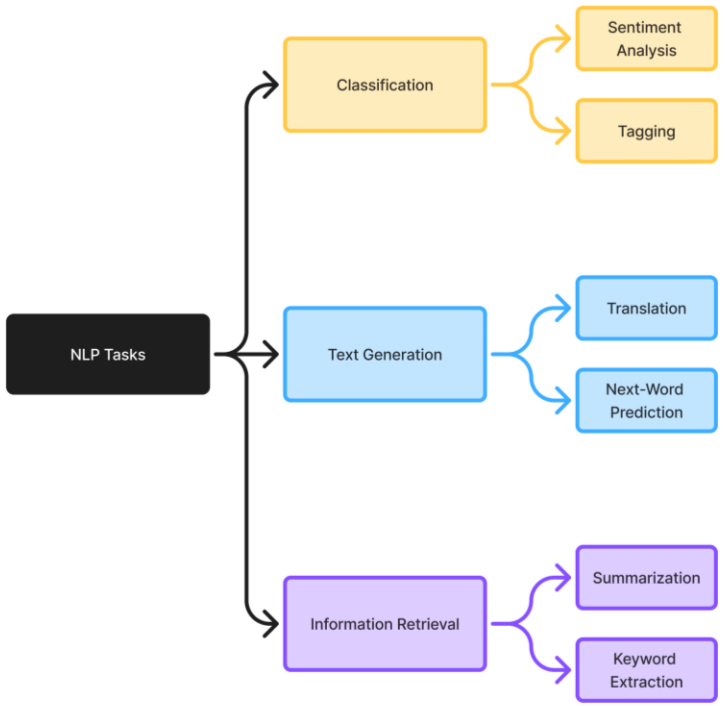
Our Proposal

Can high-resource language data be leveraged for training models in low-resource languages?



Background: Natural Language Processing

Overview



Some common NLP categories and tasks

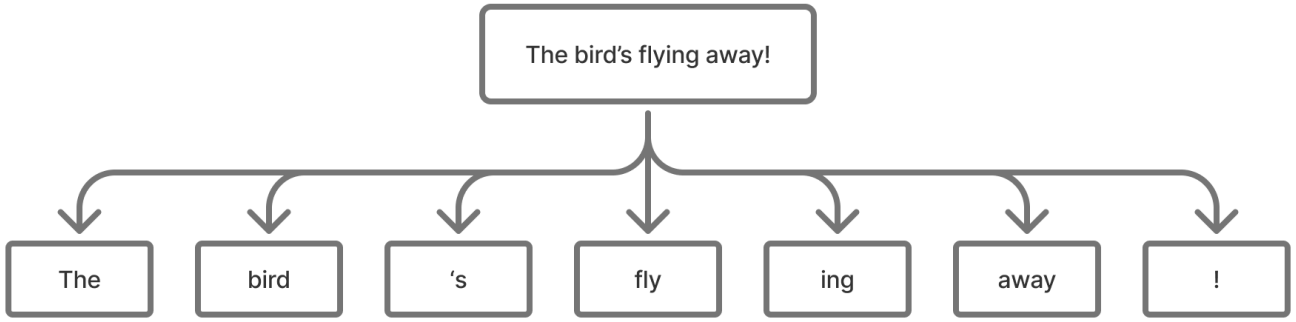
- Inputs to NLP models are variable-length strings, as opposed to set features. (Data is unstructured)
- Models learn patterns from word meaning, order, and context.

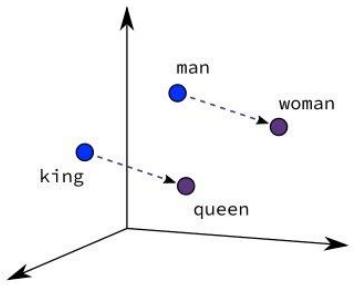
Background: Natural Language Processing

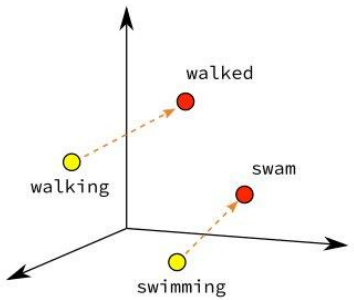
Tokenization and Vectorization

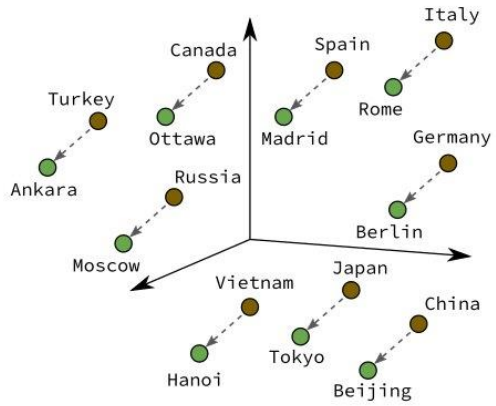
- Plaintext is divided into tokens
- Tokens are encoded into vectors
- The vector space is learned so that nearby vectors are similar
- Mathematical operations on vectors translate to words

$$vec(Madrid) - vec(Spain) + vec(France) \approx vec(Paris)$$









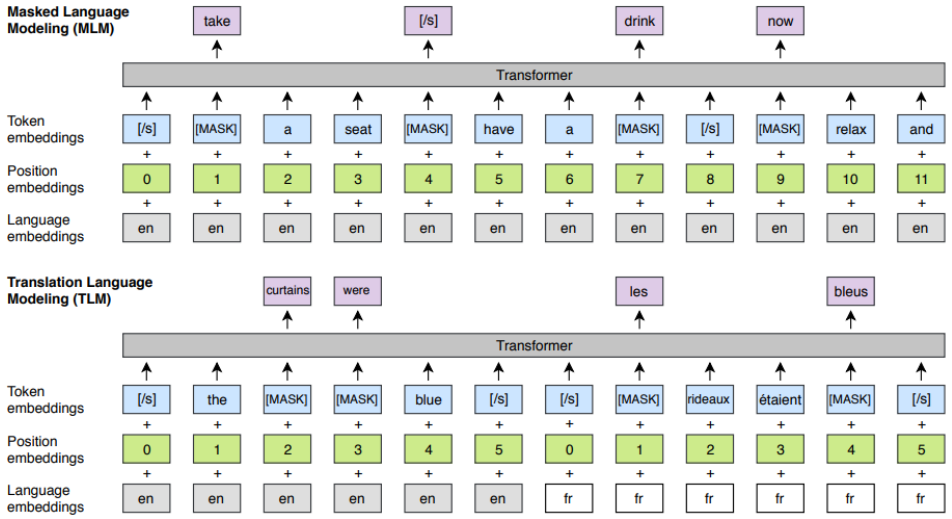
https://www.theinformationlab.nl/2023/03/22/an-introduction-to-embeddings/

Background: Natural Language Processing

BERT Architecture

Pretraining

- Model weights are tuned on unlabeled text with tasks like predicting masked words or next sentences.



Lample et. al. https://arxiv.org/pdf/1901.07291

Fine-Tuning

- Weights are 'tweaked' using labelled data for a specific task
- Task-specific 'head' layer of neurons is added on top of the base BERT model
  - The base BERT model can be thought of as a feature extractor

Dutch Baseline Model: Training from Scratch

Establish a lower bound for Dutch sentiment classification

Dataset & Tokenizer:

- 20k Dutch reviews from `procit006/dbrd_sentiment`
- Custom tokenizer created with `BertWordPieceTokenizer`
  - Vocabulary size: 30k
  - Min token frequency: 2
  - Reloaded with `BertTokenizerFast` for PyTorch compatibility

Model Architecture:

- RobBERT-style: 12 transformer blocks, 12 attention heads, Hidden size: 768
- FFN size: 3,072, No segment embeddings
- Randomly initialized weights

Training Setup:

- 5 epochs on NVIDIA P100 GPU
- Optimizer: AdamW, Scheduler: linear, no warmup, Loss: Cross-Entropy
- Metrics: Accuracy & macro F1 on validation split

Pretrained Dutch RobBERT Fine-tuning

Measure impact of language-specific pretraining

Model & Tokenizer:

- Pretrained RobBERT (`pdelobelle/robbert-v2-dutch-base`)
- Pretrained SentencePiece tokenizer and encoder weights
- Reinitialized classification head (for binary sentiment)

Training Setup (same as baseline):

- 5 epochs on NVIDIA P100 GPU
- Optimizer: AdamW
- Scheduler: linear, no warmup
- Loss: Cross-Entropy
- Metrics: Accuracy & macro F1 on validation split

Controlled Conditions:

- All hyperparameters, data, and evaluation procedures matched with baseline
- Isolates the effect of Dutch pretraining on performance

English-to-Dutch RobBERT Fine-tuning

Measure the gain of transfer learning from a similar language

Dataset:
- 100K English Amazon Book Review dataset
- 20k Dutch reviews from `procit006/dbrd_sentiment`

Model & Tokenizer:
- RobBERT trained from scratch on English
- Pretrained tokenizer and encoder weights (`bert-base-multilingual-cased`)
- Reinitialized classification head (for binary sentiment)

Training Setup (same as baseline):
- 5 epochs on NVIDIA A100 GPU
- Optimizer: AdamW
- Scheduler: linear, no warmup
- Loss: Cross-Entropy
- Metrics: Accuracy & macro F1 on validation split

Controlled Conditions:
- All hyperparameters, data, and evaluation procedures matched with baseline
- Identical Dutch sample size as in the Dutch baseline model (20K)
  - Isolates the effect of pretraining on English

German-to-Dutch RobBERT Fine-tuning

Measure the gain of transfer learning from a similar language

Dataset:
- 100k from Amazon review dataset, German subset (`mteb/amazon_reviews_multi`) (Pre-training)
- 20k Dutch reviews from `procit006/dbrd_sentiment` (Fine-tuning)

Model & Tokenizer:
- RobBERT trained from scratch on English
- Pretrained tokenizer and encoder weights (`bert-base-multilingual-cased`)
- Reinitialized classification head (for binary sentiment)

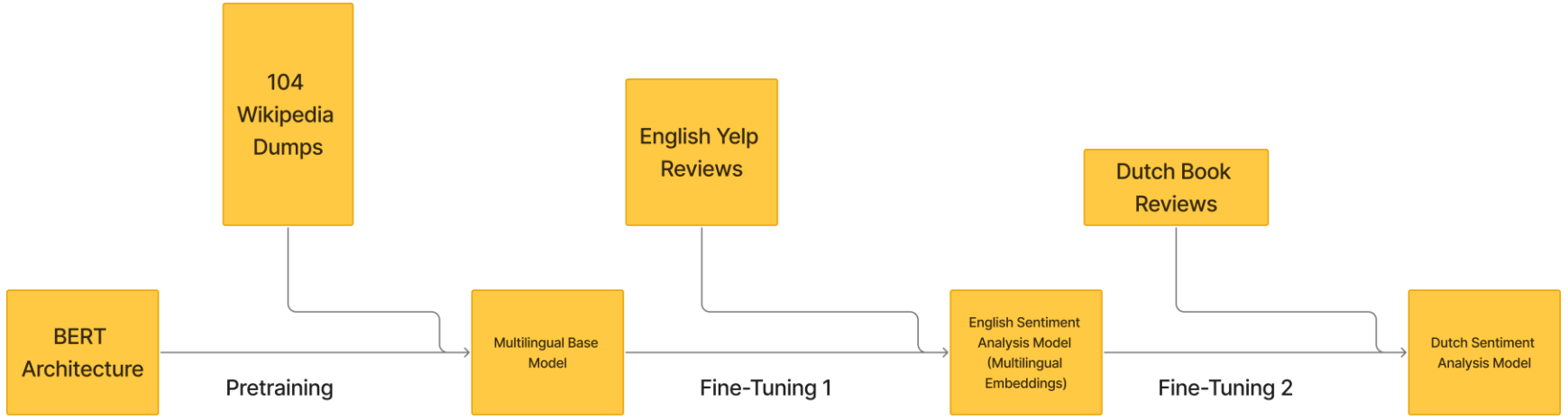
Training Setup (same as baseline):
- 5 epochs on NVIDIA RTX A550 GPUs
- Optimizer: AdamW
- Scheduler: linear, no warmup
- Loss: Cross-Entropy
- Metrics: Accuracy & macro F1 on validation split

Controlled Conditions:
- All hyperparameters, data, and evaluation procedures matched with baseline
- Identical Dutch sample size as in the Dutch baseline model (20K)
  - Isolates the effect of pretraining on German

Additional mBERT Experiments

mBERT Base Model & Double Transfer Learning

- mBERT is a **multilingual** BERT base model trained on 104 languages' Wikipedia dumps
- mBERT can be directly fine-tuned to multilingual tasks
- We tested an intermediate fine-tuning test to add more data



Additional mBERT Experiments

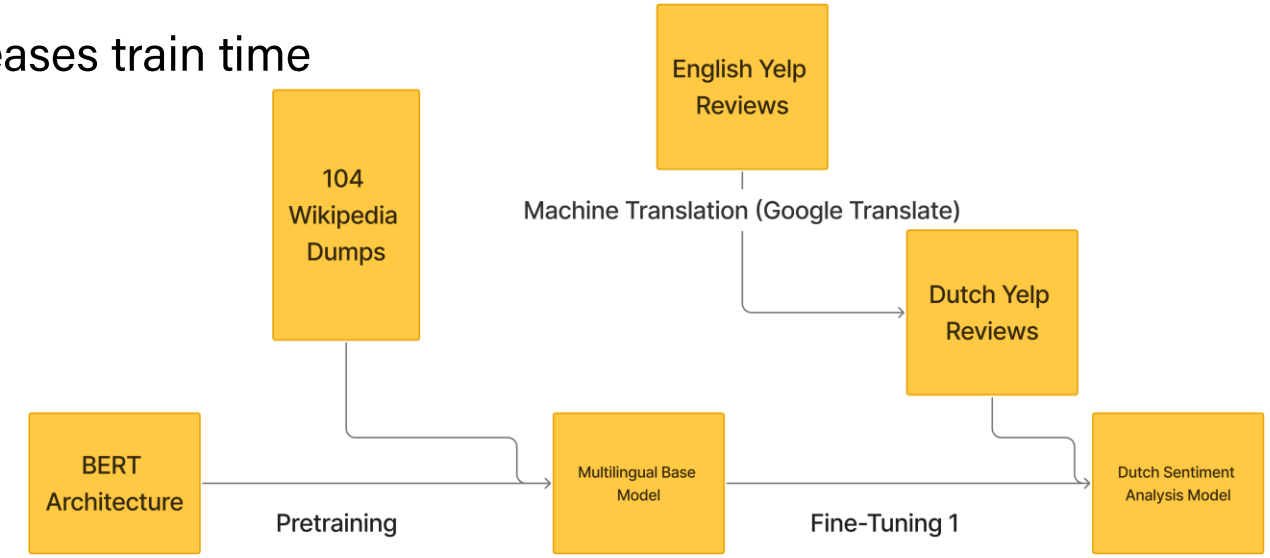
Using Machine Translation to Generate Data

Pros:

- General sentiment is likely preserved in translation
- Data for any new language can be directly fabricated from English data

Cons:

- Relies on accuracy of translation
- Large data preprocessing undertaking increases train time



Results

Model accuracies for Dutch sentiment classification

Model	Accuracy
Baseline	76.3%
Pretrained Dutch RobBERT	82.3%
German to Dutch	50.4%; 75.2%
English to Dutch	52.7%; 73.6%
mBERT	77.2%
Double Transfer Learning	66.1%
Machine Translated Data	78.9%

Discussion

- Cross-lingual transfer is more effective between **typologically similar languages**.
- mBERT's universal claim is **limited by imbalanced pretraining distributions**.
- **Monolingual pretraining remains crucial** for high-quality, language-specific NLP.

Multilingual Bias in mBERT

Why mBERT underperforms on Dutch sentiment tasks

- mBERT is pretrained on 104 languages—but English dominates the corpus
- Shared vocabulary and encoder layers tend to **optimize for English**
- Dutch input receives less aligned tokenization and representation

№	Language	Language (local)	Wiki	Articles	All pages	Edits	Admins	Users	Active users	Files	Depth
1	English	English	en	6,984,553	62,979,001	1,281,964,259	845	49,025,793	119,225	936,442	1,308
2	Cebuano	Cebuano	ceb	6,116,749	11,229,763	35,059,227	6	126,590	160	1	2
3	German	Deutsch	de	3,008,377	8,269,465	253,591,711	172	4,562,500	17,429	130,537	93
4	French	français	fr	2,678,967	13,451,082	224,536,067	143	5,161,364	17,911	74,156	269
5	Swedish	svenska	sv	2,608,820	6,306,330	57,227,585	66	946,778	1,915	0	18
6	Dutch	Nederlands	nl	2,184,962	4,699,565	69,039,923	29	1,400,647	3,607	20	19
7	Russian	русский	ru	2,040,715	8,261,383	144,374,938	63	3,739,142	8,995	259,371	162
8	Spanish	español	es	2,027,240	8,397,861	166,520,754	54	7,392,776	13,754	0	195
9	Italian	italiano	it	1,914,533	8,274,047	144,284,270	116	2,630,629	7,743	123,182	192
10	Polish	polski	pl	1,655,283	3,867,950	76,307,810	96	1,369,262	4,035	262	35

Consequences of English Bias

Multilingual models inherit the biases of their training data. “universal” encoders can become English-first by design.

- **Biased tokenization**
- **Distorted sentence embeddings**
- **Less efficient learning**
- **Reduced generalization**

Answering Our Proposal

Can high-resource language data be leveraged for training models in low-resource languages?

- **Direct English-to-Dutch transfer underperforms**
  Language gap, label shift, and syntactic mismatch hurt generalization.
- **German-to-Dutch transfer is close to the baseline**
  Language similarity can partially compensate for the lack of native-language pretraining.
- **Machine-translated English data works better**
  Dutch surface forms (even noisy) boost performance over raw English.
- **Best results come from Dutch-only pretraining (RobBERT)**
  Language-specific embeddings are still the strongest foundation.

THANK YOU

Questions?

Real Estate AI Assistant: A Data-Driven Large Language Model for Property Insights

By Amit Manchella, Dheeraj Namargomala, Parth Kulkarni, Nick Song

Motivations

Why do we care about a LLM's and what they can do?

Real Estate Complexity

Real estate involves a wide range of documents — listings, legal contracts, zoning laws, and market reports — often in inconsistent formats.

LLMs can parse and synthesize this unstructured and structured data more efficiently than traditional search methods.

Accessibility of Knowledge

First-time buyers, investors, or renters often struggle to understand processes like FHA loans, escrow, or market timing.

A conversational AI assistant can democratize real estate knowledge and provide instant, understandable answers.

Industry Inefficiencies

Agents and analysts spend hours searching documents and websites to answer common questions or gather insights.

LLMs can act as context-aware assistants, scraping webpage data and interpreting documents to speed up workflows.

Background

What is an LLM? How do they work?

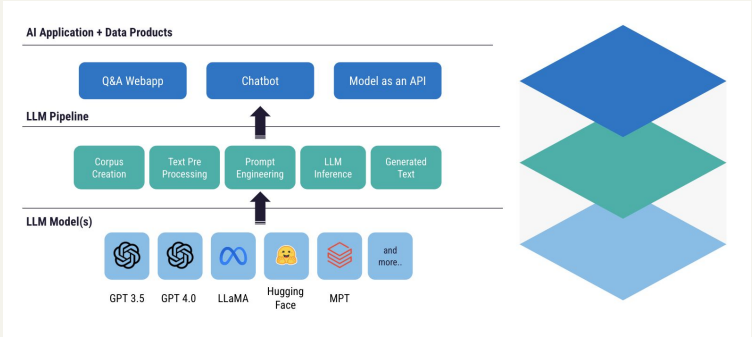
A Large Language Model (LLM) is a deep learning model trained on massive text corpora to generate, summarize, and understand natural language.

How Do LLMs Work?

- Based on **transformer architecture** (e.g., GPT, BERT)
- Learn patterns in language using attention mechanisms
- Can process prompts, retrieve context, and generate coherent responses

Why Are They Useful in Real Estate?

- Understand domain-specific jargon
- Parse varied formats (leases, listings, zoning laws)
- Answer natural language queries with contextual awareness



Current Methods

Text Mining & NLP Techniques

- Applied document parsing on Excel, PDFs, and Word files using OCR and NLP libraries (e.g., PyMuPDF, Pandas, Tika).
- Extracted key entities like property types, pricing, and location-specific terms.

Limitations:

- Messy document structures and inconsistent formatting across files.
- Requires extensive cleaning to ensure context alignment.

Our approach:

- Built a robust preprocessor pipeline using spaCy and regex-based chunking for domain-specific features.

Fine Tuning the LLM

- Full fine-tuning on domain-specific real estate data.
- Prompt engineering with curated examples.
- Retrieval-Augmented Generation (RAG) for dynamic context integration from scraped webpage content.

Limitations:

- Full fine-tuning is compute-heavy and less generalizable.
- Prompt engineering lacks dynamic document context.
- RAG can introduce latency during retrieval.

Our approach:

- We compare all three methods using standardized property-related queries for performance and efficiency.

Our Approach

Standard Methods of NLP in Real Estate

Model Architecture

- Traditional keyword search or rule-based systems

Limitations

- Poor at understanding context, nuance, and document format variability.
- Unable to generalize across different document types (e.g., leases, listings, legal docs).

Current State of the Art

Large Language Models (LLMs) + Web Contextualization

- Leverages transformer-based architectures (e.g., GPT-4) fine-tuned on real estate data.
- Incorporates webpage content scraping in real time for richer contextual grounding.
- Enables dynamic Q&A across diverse sources: Excel, PDF, Word, and HTML.

Our Model

Multi-Modal Pipeline

- **Document Loader**: Extracts raw text from Word, PDF, Excel
- **Web Context Handler**: Scrapes and parses HTML content
- **LLM Core**: Uses different fine-tuning strategies (Full fine-tuning, RAG, Prompt Engineering)

Contextual Integration Strategy

- *Full Fine-Tuning*: High accuracy on curated queries
- *RAG*: Dynamically retrieves relevant documents to reduce hallucination
- *Prompt Engineering*: Lightweight, no-training method for fast iteration

Data Collection

Data Sources

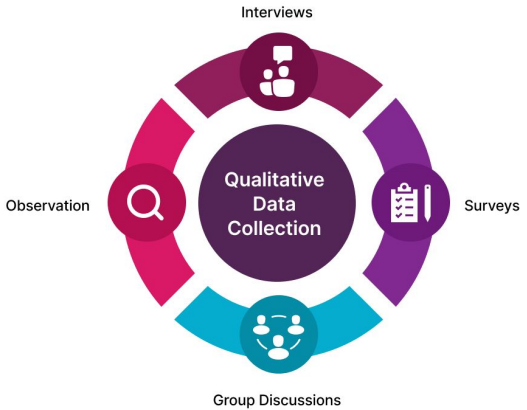
- **Structured**: Excel files with pricing, square footage, tax assessments
- **Unstructured**: PDFs and Word documents containing leases, market reports, legal agreements
- **Web**: Real estate websites scraped for listing data, school info, neighborhood trends

Tools Used

- `pandas`, `openpyxl`, `PyMuPDF`, `python-docx` for parsing
- Custom HTML scrapers using `BeautifulSoup` and `requests`

Challenges

- Non-standardized file formats
- OCR errors in scanned PDFs
- Context extraction from long documents



Fine Tuning Strategies

Strategy 1: Full Fine-Tuning

- Retrains the model weights on real estate-specific corpora
- Most accurate but expensive and rigid

Strategy 2: RAG (Retrieval-Augmented Generation)

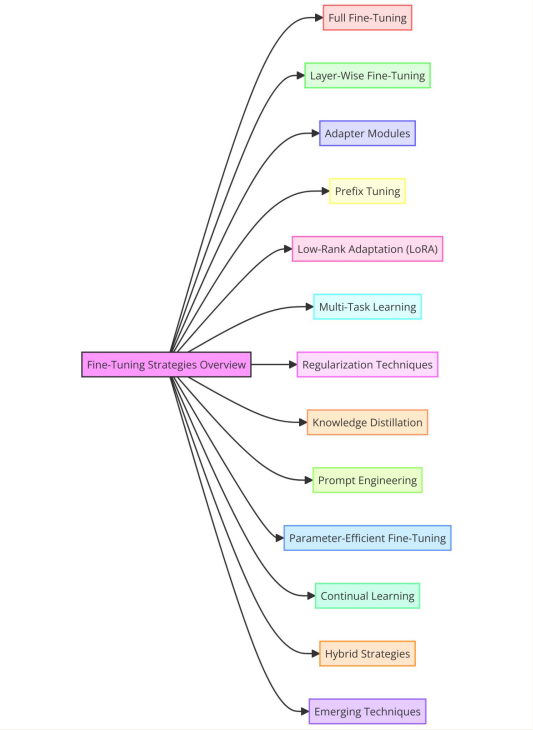
- Indexes source docs and retrieves relevant chunks at runtime
- Lightweight and dynamic — improves grounding with live data

Strategy 3: Prompt Engineering

- Curates few-shot examples in the prompt
- Fast to iterate, zero retraining needed, but sometimes less robust

Our Evaluation

- Compared based on **response relevance, latency, cost, and hallucination ra**



Results

Evaluation & Validation

Automated Metrics:

- **BLEU / ROUGE / BERTScore** – Measures how closely chatbot responses match ideal/ground truth answers.
- **Exact Match (EM)** – % of queries where chatbot returns a correct and complete answer.
- **Context Recall** – Checks whether scraped web or document context was correctly retrieved and used.

Human Evaluation:

- **Accuracy** – Was the response factually correct based on the documents?
- **Relevance** – Did it answer the question being asked?
- **Coherence & Fluency** – Did the response make logical and grammatical sense?

Advanced Analysis & Insights

- **Document Understanding**: LLM was able to parse lease agreements, extract square footage, and explain loan types.
- **Contextual Enhancement**: When scraping from webpages, the RAG model outperformed prompt engineering by 18% in accuracy.
- **Efficiency**: Prompt-engineered responses were 3× faster than full fine-tuned models.

Live Demo / Screenshots

Discussion

Smarter Real Estate Decisions

- Users gain on-demand access to complex real estate knowledge (e.g., FHA loans, rent-vs-buy analysis) in natural language.
- Reduces reliance on agents for basic information, empowering more independent and informed choices.

Workflow Automation for Professionals

- Real estate agents, brokers, and legal teams spend less time manually searching through documents.
- LLMs accelerate deal reviews, form generation, and property analysis by synthesizing information across formats and websites.

Standardization Across Markets

- By training on diverse documents and formats, the model helps normalize terminology and information representation.
- This creates a more seamless and consistent experience across cities, states, and data providers.

Improved Accessibility & Inclusion

- Many first-time buyers, immigrants, or non-experts are overwhelmed by legal jargon and scattered online resources.
- Our chatbot makes real estate knowledge more accessible to diverse populations, bridging knowledge gaps with conversational AI.

Limitations & Future Work

Limitations

- Inconsistent data formatting still causes edge-case parsing errors
- RAG performance depends on retrieval quality
- Hallucination remains a risk without domain-constrained outputs

Future Enhancements

- Train on more diverse legal and financial real estate docs
- Add multimodal inputs (images of floor plans, tables)
- Incorporate user feedback loop for continuous improvement
- Deploy in a live assistant with voice and location-based query support

Team Work and Contributions

Amit Manchella	Partth Kulkarni	Dheeraj Namargomala	Nicholas Song	Collaborative Work
Designed and implemented the core RAG chatbot architecture	Architected the Dynamic Knowledge Integration system	Built the opt-in screen scraping module for context personalization	Designed and deployed web scrapers for the initial dataset	System integration and end-to-end testing
Developed the retrieval pipeline integrating property databases and legal documents	Implemented the continuous data update pipeline with version control	Developed privacy-preserving techniques for on-screen content analysis	Created the testing framework for data quality validation	Evaluation metrics (BLEU / ROUGE / BERT Score)
Optimized the generation component for real estate-specific queries	Designed the query-driven gap detection mechanism for dataset expansion	Implemented the UI integration for real-time recommendation generation	Curated structured datasets from municipal sources and real estate APIs	Paper writing and experimental design

Conclusion

What We Built

- An LLM-powered assistant that understands real estate documents, answers questions, and contextualizes data from the web

Impact

- Bridges the gap between technical documents and user-friendly insights
- Saves hours of research for both professionals and consumers
- Enhances decision-making with instant, AI-driven responses

Summary

Our system demonstrates that fine-tuned or retrieval-augmented LLMs can significantly improve access to complex real estate knowledge.

Thank You!