# NLP: Predictive Typing System Experiment Report

Kanakorn Suk-ieam*, Khopher Sunthonkun*
* International School of Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand,
Email: 6638026521@student.chula.ac.th, 6638022021@student.chula.ac.th

*Abstract*—**Predictive typing systems enhance text input efficiency by suggesting the next word in a sequence. This paper presents an experimental comparison of various language models for a predictive typing task, specifically constrained to next-word prediction, using a large domain corpus. We evaluate models ranging from traditional statistical methods to advanced neural networks to identify the most effective approach.**

**We establish performance benchmarks using baseline N-gram models (3-gram to 6-gram) with a simple backoff strategy. To assess the impact of sophisticated smoothing, we implement a 5-gram model using the KenLM toolkit with Modified Kneser-Ney smoothing. Further, we explore neural architectures by training a custom Long Short-Term Memory (LSTM) network designed to incorporate the task's first-letter hint, and by evaluating a pre-trained GPT-2 model in a zero-shot setting using a constrained decoding strategy. Models were trained using a progressive sampling strategy on subsets ranging from 10,000 to the full 3.8 million sentences, driven by performance gains and computational constraints. All models were evaluated based on exact match accuracy on a unified development set.**

**The results demonstrate that statistical models significantly outperform the neural approaches evaluated. The 5-gram KenLM model, trained on the full dataset, achieved the highest accuracy of 63.60%, highlighting the effectiveness of Kneser-Ney smoothing combined with a large context window and extensive in-domain data. While the zero-shot GPT-2 model showed strong generalization (55.95%), it was surpassed by both KenLM and a simple 3-gram model trained on the full dataset (58.12%). The LSTM model suffered from underfitting due to data and time limitations (40.23% on 100k sentences).**

**The study concludes that for this specific predictive typing task, a well-optimized statistical model like KenLM, trained on sufficient in-domain data, provides a more accurate and computationally efficient solution than the general-purpose knowledge of pre-trained transformers or custom-trained LSTMs under the given constraints. The 5-gram KenLM model was selected for final submission.**

## I. Introduction

Predictive typing is an essential feature of modern digital keyboards, designed to improve text entry speed and user convenience. By suggesting the next word in a sentence, these systems reduce the number of keystrokes required, which is especially helpful on mobile devices. The goal of the ”Predictive Typing Contest 2” is to build such a system from the ground up. The primary challenge is to create a language model that accurately predicts the next word based on the preceding text, with the specific constraint that no spell-checking functionality is allowed. This forces the models to rely purely on the learned patterns of language.

To tackle this challenge, we explored a wide range of language modeling techniques, spanning from traditional statistical methods to state-of-the-art neural network architectures. This project aims to compare the effectiveness of these diverse approaches to identify the most suitable model for this specific predictive typing task.

We began by implementing and evaluating a series of classic N-gram models. These models work by calculating the probability of a word appearing based on the sequence of words that came before it. We tested several variations, including unigram, bigram, trigram, 4-gram, and 5-gram models, to see how much immediate context is needed for accurate predictions. We also utilized KenLM, a highly efficient and popular toolkit for building and querying N-gram models.

To move beyond the limitations of statistical methods, we then implemented two powerful neural network architectures. The first is a Long Short-Term Memory (LSTM) network, a type of recurrent neural network known for its ability to capture long-range dependencies in sequential data. The second is GPT-2, a large-scale, transformer-based language model renowned for its deep understanding of language context and its ability to generate highly coherent text.

By implementing and evaluating this broad spectrum of models—from simple N-grams to complex transformers like GPT-2—this project provides a comprehensive comparison of their performance on a pure next-word prediction task. In this paper, we will detail the implementation of each model, analyze their respective strengths and weaknesses, and present our findings on which approach yields the most accurate predictions for a simulated predictive keyboard.

## II. Experiment Setup

### A. Dataset

In this contest, the given dataset is separated into 3 files: training set, development set, and testing set.

*1) Training Set:* The training corpus is extensive, consisting of approximately 3.8 million sentences, which totals over 128 million tokens. This large scale provides a rich foundation for learning complex linguistic patterns. The vocabulary size is 99,021 unique tokens. The dataset exhibits high quality and density, as indicated by the low frequency of rare words. Only 8.31% of the vocabulary (8,233 tokens) appears five times or fewer, and there are only three singleton tokens (words appearing once). This robust distribution ensures that the statistical properties learned by the models are reliable, even for less common words.

Tokenization of the corpus follows the Penn Treebank style, where punctuation marks like periods (.), commas (,), and hyphens (-) are treated as distinct tokens. A notable characteristic is the handling of contractions; for example,

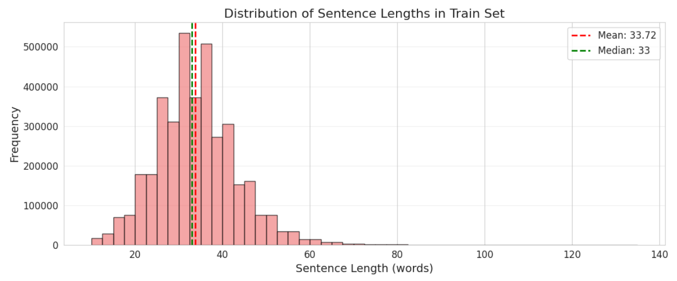

Fig. 1. Distribution of Context Lengths of the Training Set

”it’s” is tokenized into three separate tokens: it, ’, and s. An analysis of token frequencies (Table 1) shows that the most common tokens are dominated by stopwords (the, a, of) and punctuation. Furthermore, the high frequency of specific numbers (11, 1) and days of the week (tuesday, wednesday) suggests the presence of domain-specific content, likely from news or financial reports.

*2) Development Set:* The development (evaluation) set contains 94,825 instances, each requiring a single next-word prediction. Each instance consists of a context sequence and the first letter of the target word. The provided contexts are sufficiently long to support models that capture long-range dependencies, with a mean length of 24.95 words and 95% of contexts being 37 words or shorter.

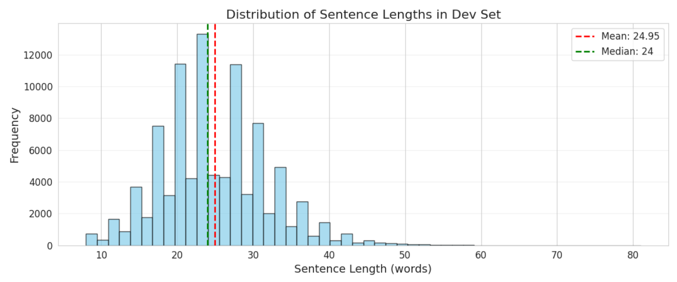

Fig. 2. Distribution of Context Lengths of the Development Set

The primary constraint of the task is that the prediction must begin with the given first letter. The distribution of these initial letters is varied. While t (13.64%), a (10.46%), and s (7.12%) are the most common alphabetic starting letters, a significant challenge is posed by non-alphabetic targets. Approximately 14.3% of all required predictions are punctuation marks, numbers, or symbols. The most common non-alphabetic targets are the comma („ 4.98%), the hyphen (-, 2.38%), and the number 1 (2.20%). Consequently, a successful model must not only predict conventional words but also learn the syntactic rules governing the placement of punctuation and other special characters.

## B. Evaluation Metrics

Model performance is measured by accuracy. A prediction is considered correct only if it exactly matches the target word in the development set.

Crucially, all model were evaluated on the same development set. This unified evaluation ensures that all reported accuracy metrics are directly and fairly comparable.

## C. Training Methodology and Sampling Strategy

Given the large 3.8-million-sentence training corpus and the limited computational resources, we adopted a progressive sampling strategy. This approach was essential to balance model exploration with computational efficiency, particularly for slow-training neural models.

*1) Sampling Subsets:* We created four distinct, progressively larger training subsets by taking the first N sentences from the training file:

- **10,000 sentences**: For quick prototyping and syntax validation.
- **100,000 sentences**: For early performance assessment.
- **1,000,000 sentences**: For scaled training for promising models.
- **3,803,957 sentences**: For final training of efficient, high-performing models.

An exploratory analysis confirmed that sentences in the corpus are independent excerpts. This allowed the use of sequential sampling, which is statistically equivalent to random sampling for this use case while offering superior reproducibility and efficiency.

*2) Criteria for Scaling:* A cost-benefit analysis was employed to determine whether to train a model on the next-largest dataset size. The decision to scale up was based on two criteria:

- **Performance Improvement**: We evaluated gains on two fronts: an accuracy improvement of at least 5 percentage points (with an exception of trigram, as it is used as a baseline) and the *relative* gain compared to the smaller models. If a model showed diminishing returns (i.e., a much larger model yielded only a marginal improvement), it was deprioritized for training on the full corpus.
- **Training Time**: Practical time limits were set (e.g., ¡ 2 hours per epoch for neural models). If an epoch’s estimated time exceeded this, scaling was reconsidered.

## III. Models

To address the problem, we explored a range of language models, progressing from traditional statistical methods to state-of-the-art neural network architectures. This progression allowed us to establish a strong baseline and evaluate the benefits of more complex models with greater contextual understanding. This section details the motivation, architecture, and hyperparameter tuning process for each model attempted.

N-gram models served as our initial baseline due to their simplicity, interpretability, and computational efficiency. These statistical models work by estimating the probability of a word given the previous $n-1$ words. We hypothesized that by experimenting with different context window sizes (i.e., different values of $n$), we could identify the point at which the benefits of longer context were outweighed by issues of data sparsity. We implemented several n-gram models, specifically for $n = 3$ (trigram), $n = 4$ (4-gram), $n = 5$ (5-gram), and $n = 6$ (6-gram). The key hyperparameter for these models was n (the context window size). We systematically trained and evaluated models for n = 3, 4, 5, and 6 to observe the performance trade-offs between longer context and increased data sparsity. The design decisions for our models included:

- Sentence Boundaries: Each sentence in the training corpus was prepended with start-of-sentence tokens (<s>) and appended with an end-of-sentence token (</s>). This ensures that n-gram contexts do not cross sentence boundaries, which would be grammatically meaningless.
- Backoff Strategy: To handle the problem of data sparsity (where a specific n-gram has not been seen in the training data), we implemented a recursive backoff strategy. If a given n-gram's probability could not be calculated, the model would fall back to the shorter $(n-1)$-gram model. For instance, the trigram model would back off to a bigram model, which would in turn back off to a unigram model.
- First-Letter Fallback: As a final step in the backoff chain, if a word could not be predicted based on unigram probabilities, the model defaulted to selecting the most frequent word in the entire vocabulary that matched the required first letter.

While our custom n-gram models provided a solid baseline, we sought to compare them against a highly optimized, industry-standard implementation. KenLM is a popular toolkit for language modeling that incorporates more sophisticated smoothing techniques than simple backoff. We specifically chose KenLM to leverage its implementation of Modified Kneser-Ney smoothing, which is known to be more effective at handling unseen n-grams and distributing probability mass. Similar to our n-gram models, the main hyperparameter for KenLM was the n-gram order (n). We again tested values from 3 to 6 to maintain a direct comparison. The decision to use Kneser-Ney smoothing was based on its established and highly effective technique for language modeling. The final value for n was decided by comparing the performance of each model on our validation set and selecting the one that achieved the highest accuracy.

To overcome the fixed-context limitations of n-gram models, we implemented a Recurrent Neural Network (RNN) using a Long Short-Term Memory (LSTM) architecture, which is capable of capturing long-range dependencies in text. Our model features a novel design to explicitly handle the task's first-letter constraint. The architecture processes the word context through a multi-layer LSTM to generate a context-aware summary vector. In parallel, the required first letter is converted into a dense vector via a separate embedding layer. These two vectors—representing the preceding context and the future constraint—are then concatenated. This combined feature vector is passed through a final linear layer to produce a probability distribution over the vocabulary, ensuring the prediction is conditioned on both the semantic context and the required starting character. We optimized the model's performance by tuning several key hyperparameters based on the cross-entropy loss on a held-out validation set. This search included the dimensionality of the word embeddings (embedding_dim), the size of the LSTM's hidden layers (hidden_dim), the model's depth (n_layers), and the dropout probability (drop_prob) for regularization. The learning rate for the Adam optimizer was also carefully selected to ensure stable convergence. We optimized the model's performance through grid search and manual adjustments guided by performance on validation set. The final configuration was chosen by selecting the combination of hyperparameters that yielded the lowest cross-entropy loss. The specific parameters we tuned were:

- embedding_dim: The dimensionality of the word embeddings. We tested values of 64, 128, and **256** to find the right balance between representational capacity and model complexity.
- hidden_dim: The size of the LSTM's hidden layers. We explored 128, 256, and **512** units to control the model's learning capacity.
- n_layers: The model's depth. We experimented with stacking 1, **2**, and 3 LSTM layers to see if deeper models could learn more complex patterns.
- drop_prob: The dropout probability for regularization. We tested rates of 0.3, **0.4**, and 0.5 to prevent overfitting.
- Learning Rate: Using the Adam optimizer, we searched for an optimal learning rate to ensure stable and efficient convergence.

Notes: The bold values are the selected hyperparameters.

Our final approach involved a pre-trained GPT-2 model, leveraging its powerful, generalized understanding of language without the computational cost of fine-tuning. The core of this method was an efficient, inference-time constrained decoding strategy. We used the standard gpt2 model to generate a probability distribution for the next word given a context. From this distribution, we identified the top_k most likely candidate tokens.

This list of high-probability candidates was then filtered in a post-processing step. We decoded each candidate token and selected the first one in the probability-ranked list that satisfied our task's specific first-letter constraint. This method effectively guides the transformer model's predictions to a valid and contextually appropriate answer. The only hyperparameter tuned for this approach was top_k, which defines the size of the candidate pool. We set k=1000 to ensure a sufficiently large pool for finding a matching word without sacrificing efficiency.

## IV. Results

Table I presents the comparative performance for each model implemented, categorized by the size of the training dataset used. The primary metric for comparison is the **exact match accuracy** on the full development set, as defined in our methodology.

Our experiments reveal two primary finding. Firstly, **statistical models with smoothing (KenLM) outperformed all other architectures**, including neural models. Secondly,

TABLE I
OVERALL MODEL PERFORMANCE (DEV ACCURACY %)

| Training Sentences | Simple 3-gram | Simple 4-gram | Simple 5-gram | Simple 6-gram | KenLM (5-gram) | LSTM | GPT-2 |
|---|---|---|---|---|---|---|---|
| **10,000** (Debug) | 43.42% | 43.54% | 43.79% | 43.78% | 46.21% | 31.47% | — |
| **100,000** (Dev) | 49.89% | 50.41% | **50.73%** | 50.68% | 52.80% | 40.23% | — |
| **1,000,000** (Large) | 54.08% | 55.23% | 55.67% | — | 57.30% | — | — |
| **3,803,957** (Full) | 58.12% | — | — | — | **63.60%** | — | — |
| pre-trained | — | — | — | — | — | — | 55.95% |

*Note: "—" indicates a combination that was not run due to computational or time constraints, with an exception of pre-trained, as it's only applicable to GPT-2*

**training data size was the single most important factor** for improving the accuracy of any given model.

## A. KenLM with smoothing

The 5-gram model built with KenLM, which utilizes Kneser-Ney smoothing, achieved the highest final accuracy of **63.60%**. This success is attributed to its ability to handle data sparsity: it leverages a large 5-gram context when available but "backs off" intelligently to smaller n-grams without the steep probability penalties of a simple count-based model.

## B. Simple N-grams:

These models acted as strong baselines. The 3-gram model, when scaled to the full 3.8M sentence dataset, achieved an impressive **58.12% accuracy**. This "brute force" approach (simpler model, more data) ultimately outperformed more complex but data-starved models, such as the 5-gram model trained on 1,000,000 sentences (55.67%).

## C. LSTM

The LSTM model performed poorly, achieving only **40.23%** accuracy with 100k sentences. This was worse than even the 10k-sentence 3-gram model (43.42%), which suggests the model was severely **underfit**. LSTMs require vast amounts of data and long training times to learn complex dependencies. However, training with full training could take significantly longer than other models, so we decided to deprioritize it after implementing with 100k sentences.

## D. GPT-2

The pre-trained GPT-2 model was evaluated in a **zero-shot** setting, meaning it was **not trained or fine-tuned** on any of the 3,803,957 sentences from our training set. Even without seeing any in-domain data, it achieved a strong accuracy of **55.95%**. This result significantly outperformed our trained LSTM model (40.23%), demonstrating the powerful generalization capabilities of a large pre-trained Transformer.

However, it was ultimately less effective than other smaller models that can learn directly from the training set. It was outperformed by both the **KenLM model (63.60%)** and the simple **3-gram model (58.12%)** trained on full training dataset.

## E. The Impact of Data Scaling

For every model tested (excluding pretrained GPT-2), increasing the training data resulted in a significant, predictable accuracy boost. For instance, KenLM achieved 46.21% with 10,000 samples, 57.30% with 1,000,000 samples, 63.60% with the full training set.

This trend confirms that for language modeling, data is the one of the most critical resource. The statistical models were simply more computationally efficient, which allowed us to train them on more training samples and unlock their performance.

## V. CONCLUSION

From our experimental results, we find that statistical language models, specifically those with sophisticated smoothing techniques, decisively outperform both custom-trained recurrent neural networks (LSTMs) and pre-trained transformers (GPT-2) on this specific next-word prediction task.

Among all architectures, the **5-gram KenLM model** achieved the most robust overall performance. Its success is attributed to its ability to balance two key factors: it leverages a large context window (5-gram) while using Kneser-Ney smoothing to effectively handle data sparsity. This combination allowed it to leverage the full 3.8-million-sentence training set more effectively than any other model.

Although the zero-shot GPT-2 model (55.95%) demonstrated powerful generalization and the simple n-gram models showed the impressive utility of a "brute force" data-driven approach, neither could match the specialized accuracy of the smoothed KenLM model. This study demonstrates that for a constrained predictive typing task, a well-optimized statistical model trained on sufficient in-domain data remains a more effective and computationally efficient solution than more complex, general-purpose neural architectures.

For our final submission to the contest, we selected the test-set results generated by the **5-gram KenLM model trained on the full 3,803,957-sentence dataset**, as it provided the highest accuracy on the development set.