Pinyin-to-Hanzi Converter with N-gram Language Model

Lux Zhang

Problem

- Modern Chinese writing uses over 8,000 unique characters
- Most Chinese speakers use only alphanumeric keys to type (qwerty)
- Problem?

“nihaoshijie” → “你好世界”

Typed **pinyin** (reading)

Hanzi (characters) output

O(40^n) possibilities

- Solutions for typing Chinese *(user-interactive)*: **IMEs (Input Method Editors)**
  - Requires user intervention!

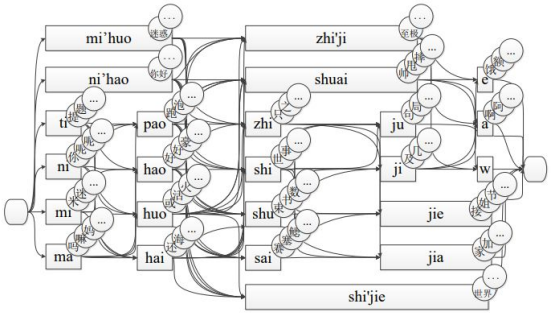
Background

- IMEs have been in use since some OS implemented non-Latin orthography
  - Microsoft Windows 3.0 first implemented multilingual support
  - Early IMEs are often inaccurate and limited in vocabulary due to memory restrictions
- **Rime**: open-source modern IME for Chinese
  - Uses frequency of dictionary entries and ad-hoc heuristics
  - Data in N-gram format, but only contains dictionary entries
  - Small corpus size: 204 million segmented words

RIME

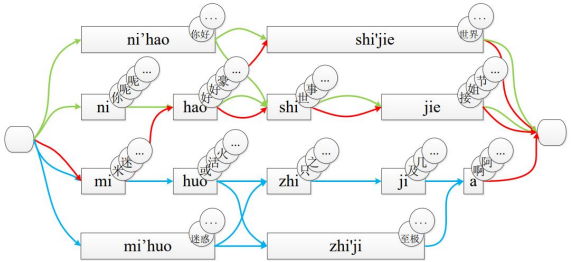
Background (cont'd)

- **Joint Graph model** for pinyin-to-Chinese conversion [JZ14]
  - Uses a Hidden Markov Model
  - Typo correction (K-shortest paths)



Joint Graph model

K-shortest paths



Filtered Joint Graph model

38% sentence accuracy, 96.24% character accuracy

Project Statement

Design, implement, and evaluate:

Non-interactive pinyin-to-hanzi converter for Chinese

- Newly-built N-gram language model w/ beam search
- ~93% character-by-character accuracy
- I chose to take a simple approach
  - Rather than creating a model with even more assumptions...
  - Operate directly from character data to build the model
  - Simple model w/ large memory footprint vs. smaller, more complex system

System Overview

- **N-gram table:** an unordered map containing an in-memory data structure representation of individual N-grams
- **Pinyin-to-char map**: a mapping consisting of one pinyin reading and all possible hanzi associated with it
- **Pinyin Moistener**: tokenize pinyin input, get all possible hanzi from map, beam search through N-grams

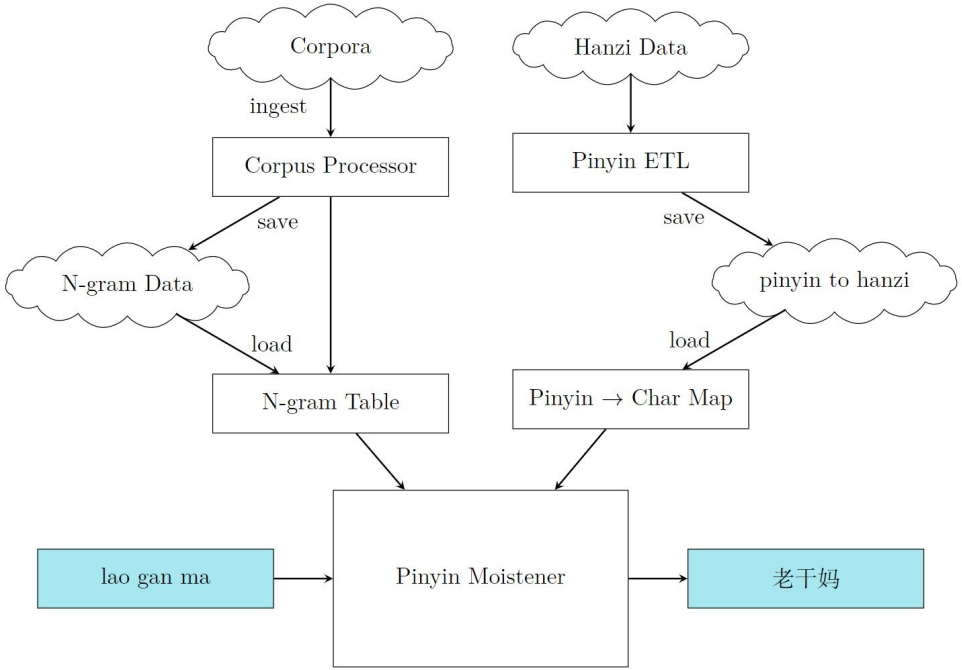
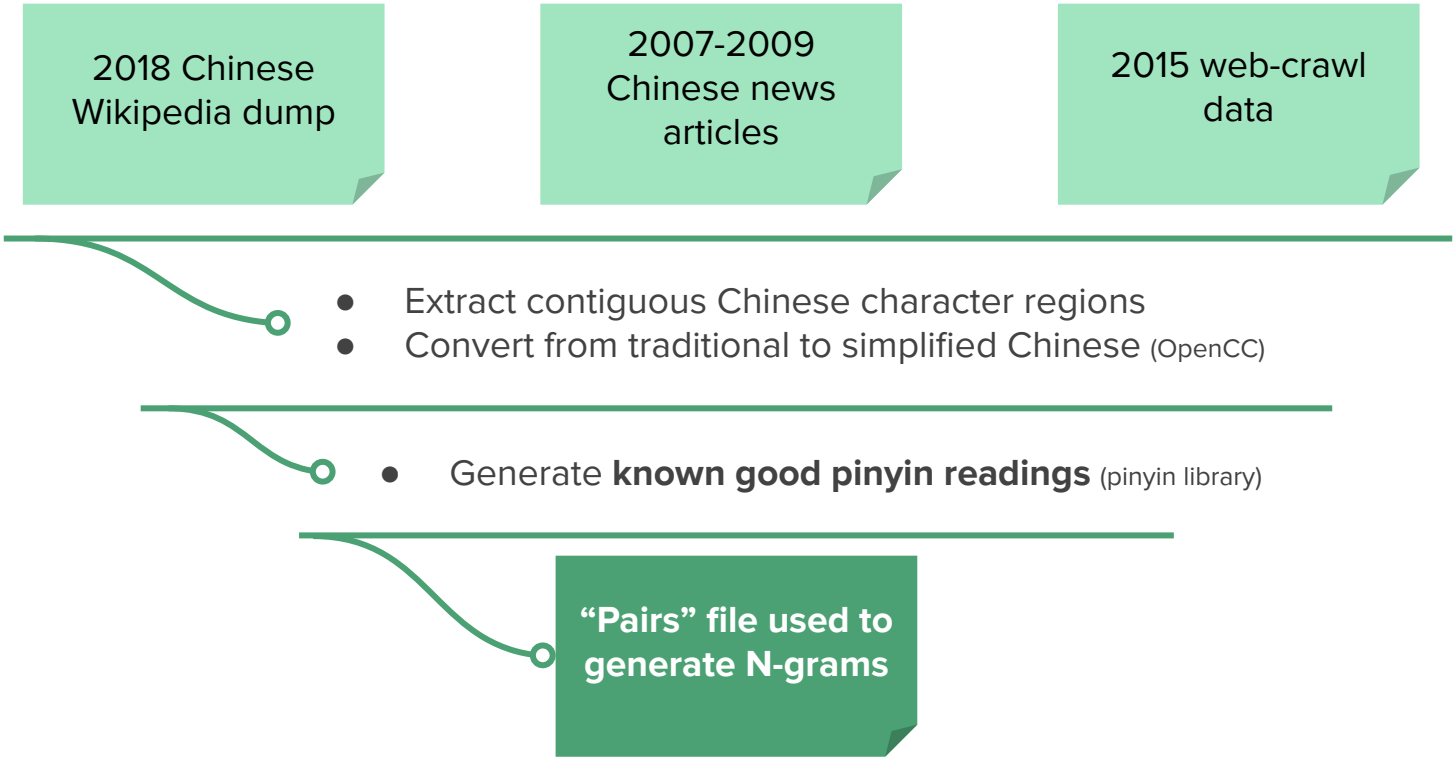


Figure 1. Block diagram showing the overall system.

Data Sourcing



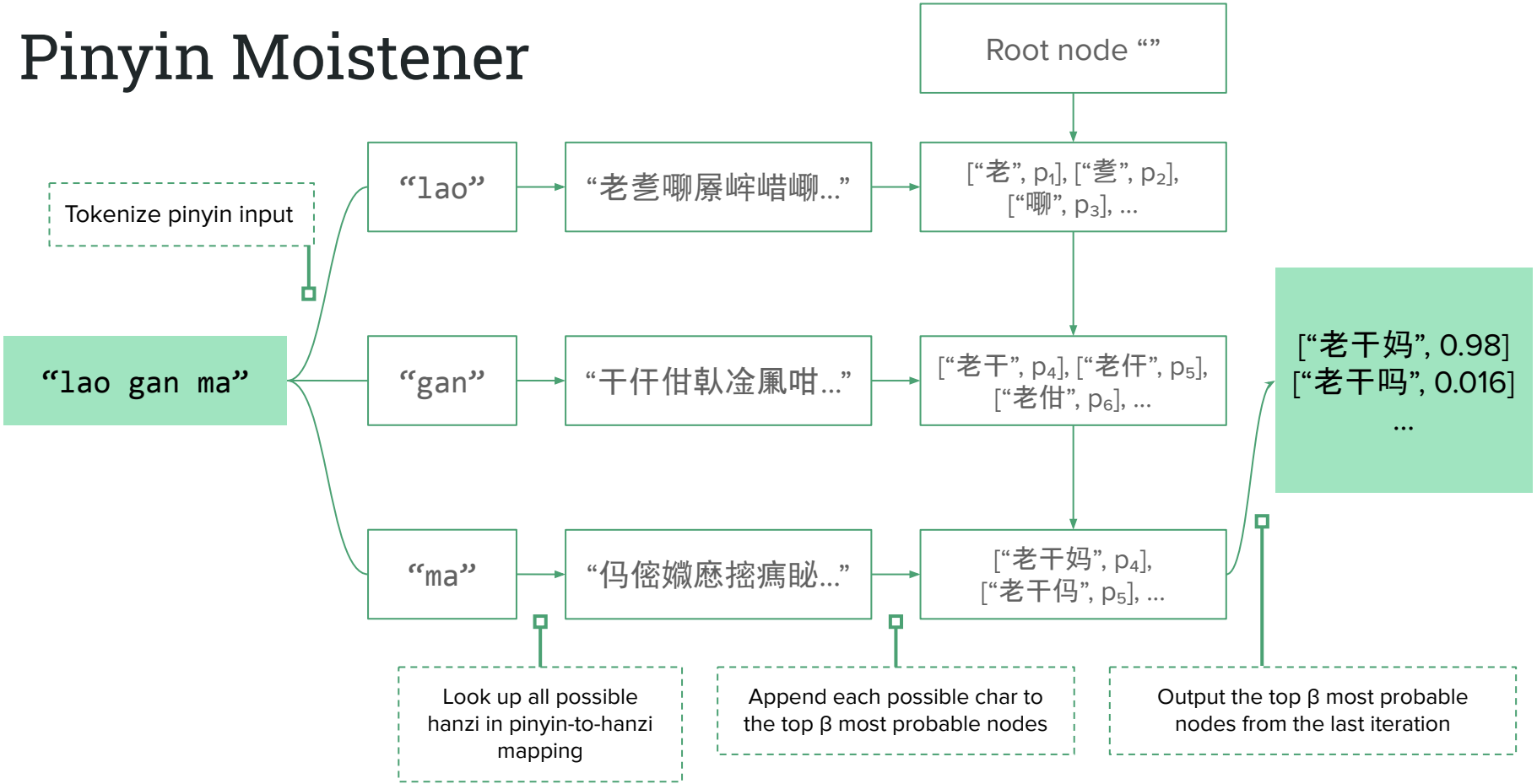
N-Gram Generation

- Statistical noise presents a challenge for approximating the probability of some N-grams
  - No segmentation means that the model will ingest N-grams that cross word boundaries
    - Include grammatical particles
    - Might only occur once
- **Smoothing**
  - Approximates the probability of character c_n following characters $c_1, \ldots, c_{n-1}$ as

$$Pr(c_n|c_1c_2\ldots c_{n-1}) \approx \frac{\epsilon_1 + \#(c_1c_2\ldots c_n)}{\epsilon_2 + \#(c_1c_2\ldots c_{n-1})}$$

where $\#(s)$ indicates the number of times our model has seen s. The epsilon values then become hyperparameters for our model.

Pinyin Moistener

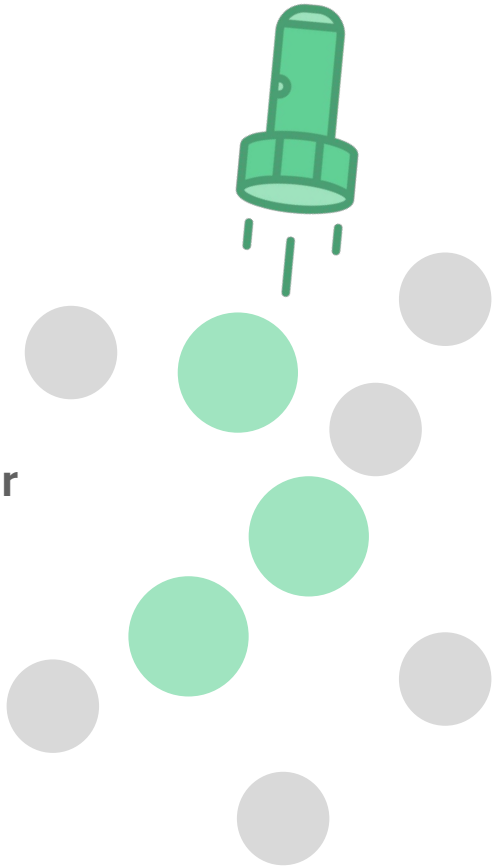


Beam Search

- Exhaustive search algorithms are optimal, but…
  - Expands upon every child node
  - Branching factor tied to N-gram order
- Beam search: only top β nodes are expanded upon
  - Suboptimal
  - Reduces memory usage significantly
- In its essence, breadth-first search with **pruned frontier** of size β (**beam width**)
  - In each iteration, pick top β results ranked by the probability

$$Pr(c_n|c_1c_2\dots c_{n-1})$$

  - Beam width is a hyperparameter



Performance Analysis: Beam Width, N-gram Order

- **Beam width** (Table 1)
  - Drastic improvement until β = 50
  - Slows down due to diminishing returns
  - No improvement after β = 100
- **N-gram order** (Table 2)
  - 1-, 2-, 3-grams all show significant improvement
  - 1-2% increase in accuracy metrics from 3- to 4-grams not worth more than doubled memory usage

Width	Top-1 Acc	Top-5 Acc	Top-10 Acc	Char Acc	Time (ms)
5	58.0726	63.4543	63.4543	86.8209	3.94985
10	68.0851	76.4706	76.9712	90.5674	7.8149
20	72.0901	82.9787	84.2303	91.8925	13.0297
30	72.3404	84.6058	85.9825	92.1817	20.09
50	73.2165	85.8573	87.4844	92.3503	31.8671
75	73.3417	86.4831	88.6108	92.4949	45.1071
100	73.592	86.7334	88.8611	92.5431	58.2917
150	73.592	86.7334	88.8611	92.4828	84.5306
200	73.592	86.7334	88.9862	92.519	114.052

Table 1. Performance metrics by beam width.

Order	Top-1 Acc	Top-5 Acc	Top-10 Acc	Char Acc	Time (ms)	Memory
1-grams	3.87985	12.015	16.3955	45.115	30.8722	0.01 GB
2-grams	46.3079	68.9612	74.8436	85.05	46.1863	0.2 GB
3-grams	73.592	86.7334	88.8611	92.5431	58.2917	2 GB
4-grams	76.3454	87.1089	89.3617	93.0008	98.4544	5 GB

Table 2. Performance metrics by N-gram order with beam width 100.

Performance Analysis: Corpus Size

- **Corpus size** (Table 3)
  - This parameter is very important to this model because my approach does not rely on word/phrase dictionaries and other heuristics
  - Drastic improvement from 30,000 to 1,500,000 sentences
  - Smaller but still significant improvement from 1,500,000 to 2,700,000 sentences

Corpus Size	Top-1 Acc	Top-5 Acc	Top-10 Acc	Char Acc	Time (ms)
30,000	47.6846	65.2065	69.587	85.4355	51.7973
300,000	63.7046	77.3467	80.1001	89.8085	52.0118
1,500,000	70.8385	84.3554	87.1089	91.7118	61.4025
2,700,000	73.592	86.7334	88.8611	92.5431	58.2917

Table 3. Performance by corpus size in sentences with beam width 100.

Performance Analysis: Genre Transfer

- **Genre Transfer** (Table 4)
  - 900K sentences are extracted from each individual corpus as well as all corpora combined
  - Test dataset from all corpora
  - Web corpora outperforms other two genres, which see decreases in accuracy metrics
    - News and wiki corpora might contain more formal/professional style writing
    - Web contains a mix of different styles of writing, making it better suited for general test dataset

Genre	Top-1 Acc	Top-5 Acc	Top-10 Acc	Char Acc	Time (ms)
900K All	69.3367	82.1026	85.2315	91.3745	55.7231
900K Web	73.4668	85.2315	87.234	92.2419	59.2902
900K News	49.562	66.2078	70.9637	86.7365	51.264
900K Wiki	43.4293	59.199	62.5782	82.0744	52.1054

Table 4. Performance by training data genre with beam width 100.

...man!

Demo!