GeoIndia: A Seq2Seq Geocoding Approach for Indian Addresses

Bhavuk Singhal
Meesho
bhavuk.singhal@meesho.com

Anshu Aditya
Meesho
anshu.aditya@meesho.com

Lokesh Todwal
Meesho
lokesh.todwal@meesho.com

Shubham Jain
Meesho
shubham.jain1@meesho.com

Debashis Mukherjee
Meesho
debashis.mukherjee@meesho.com

Abstract

Geocoding, the conversion of unstructured geographic text into structured spatial data, is essential for logistics, urban planning, and location-based services. Indian addresses with their diverse languages, scripts, and formats present significant challenges that existing geocoding methods often fail to address, particularly at fine-grained resolutions. In this paper, we propose ***GeoIndia***, a novel geocoding system designed specifically for Indian addresses using hierarchical H3[1]-cell prediction within a Seq2Seq framework. Our methodology includes a comprehensive analysis of Indian addressing systems, leading to the development of a data correction strategy that enhances prediction accuracy. We investigate two model architectures, Flan-T5-base (T5) (Chung et al., 2024) and Llama-3-8b (QLF-Llama-3) (Meta), due to their strong sequence generation capabilities. We trained around 29 models with one dedicated to each state, and results show that our approach provides superior accuracy and reliability across multiple Indian states, outperforming the well-renowned geocoding platform ***Google Maps***[2]. In multiple states, we achieved more than an ***50%*** reduction in mean distance error and more than a ***85%*** reduction in 99th percentile distance error compared to Google Maps. This advancement can help in optimizing logistics in the e-commerce sector, reducing delivery failures and improving customer satisfaction.

1 Introduction

The rise of e-commerce has transformed the way we shop, offering unmatched convenience and a vast array of products at our fingertips. However, the smooth online shopping experience relies on a complex logistics network that ensures timely, efficient, and reliable delivery. Geocoding plays a crucial role in this process. It helps in planning the logistics network, from selecting facility locations and assigning hubs or distribution centers to planning delivery routes. By ensuring timely deliveries, geocoding not only enhances the customer experience but also reduces failed deliveries and RTO[3], a financial threat to the e-commerce industry.

Building a geocoder for Indian addresses presents unique challenges not typically encountered in other countries. Unlike western countries with standardized addressing systems, Indian addresses are highly diverse and lack uniformity. For example: In contrast to the simplicity of addresses like *"10 Downing Street"* in the UK, Indian addresses can be significantly more complex. Consider the example: *"XX[4], palasi mohania village madarsa chowk rto jama masjid Araria Bihar, 854333, purnia"*. Here, it is challenging to discern that *"palasi mohania"* is intended as a locality, *"madarsa chowk"* and *"rto jama masjid"* as potential landmarks, *"araria"* as the municipality within the city of *"purnia"*, and *"854333"* as the postal code. This complexity is compounded by the use of various regional terms like *street*, *main/cross*, *colony*, and more, which vary significantly among states and regions. The traditional use of addresses in India was relatively informal until the advent of e-commerce in the early 2000s. Previously, addresses were shared verbally or handwritten for postal deliveries, often including landmarks to guide the recipient. However, entering these addresses into digital systems for online shopping has highlighted the inconsistencies inherent in Indian addresses.

Several factors contribute to the difficulty of geocoding in India:

[1]https://www.uber.com/en-IN/blog/h3/
[2]https://www.google.com/maps

[3]Return to Origin (RTO) refers to the non deliverability of a package to the buyer and its return to the sellers address
[4]Due to business confidentiality, some exact values are not revealed, and finer address details are masked (XX) to ensure the privacy of customers.

Lack of Standardization: There is no standardized format for addresses across India. Each state and even different regions within states use varied terminologies and structures, making it challenging to develop a one-size-fits-all geocoding solution. For example, an address in Mumbai might be written as *"Flat No. XX, Building No. 5, XX Society, Andheri West, Mumbai, Maharashtra, 400053"* which includes specific details about the building and locality. In contrast, an address in a rural area of Tamil Nadu might be *"House No. XX, Near Big Temple, Thanjavur District"* which relies more on prominent local landmarks.

Inconsistent Address Formatting: Addresses may be written in various orders, with elements like the state, pincode, or house number appearing in different sequences. For example, an address in Chennai might be written as *"No. XX, 2nd Cross Street, Besant Nagar, Chennai - 600090, Tamil Nadu"* while another in the same city could be *"Besant Nagar, XX, 2nd Cross Street, Chennai - 600090, Tamil Nadu"*.

Incomplete and Erratic Data: Addresses often lack key components or contain errors, such as misspellings and incorrect locality or street names. For example, the locality *"daharahara chawk"* can be misspelled as *"dharhara chowk"*.

Reliance on Landmarks: Many addresses in India use informal descriptions and landmarks, like *"near the big banyan tree"* or *"behind the supermarket"* which are not standardized and can be ambiguous. This is especially prevalent in rural areas, making them difficult to incorporate into a geocoding model.

Traditionally, geocoding has been addressed using rule-based and heuristic strategies. However, recently, Kothari et al. (Kothari and Sohoney, 2022) and Reddy et al. (Reddy et al., 2022) formulated geocoding as a Seq2Seq task, where coordinates are converted into grids and predicted in an autoregressive manner. Building on this research, we have conducted empirical evaluations, incorporating the nuances of Indian addresses and scaling the solution across all 29 Indian states using low-latency and high-throughput model serving infrastructure. The major contributions of this work are:

1. We explored different language model backbones, specifically Flan-T5 and Llama-3, in the context of Indian address geocoding.
2. We demonstrated the benefits of transfer learning and domain specific tokenization in scaling the model to multiple regions with improvement in accuracy and convergence time.
3. We examined the effect of varying batch sizes and gradient accumulation steps in data-scarce settings, optimizing for both accuracy and efficiency.
4. We showcased the practical effectiveness of our pipeline across each Indian states and comparing our performance with Google Maps.

2 Related Work

The evolution of geocoding techniques has transitioned from traditional rule-based and heuristic strategies to sophisticated deep learning models.

Initial geocoding research used rule-based and traditional machine learning methods that mapped text to geographic locations by extracting and ranking entries from address databases (Zhang and Gelernter, 2014; Karimzadeh et al., 2019; Viegas, 2021; Karimzadeh et al., 2013; Lieberman and Samet, 2012). Although these methods worked well, they faced challenges in regions without extensive, high-quality databases, limiting their effectiveness in areas with sparse or non-standardized address data (Goldberg et al., 2007; DeLozier et al., 2015; Kulkarni et al., 2020).

To overcome the limitations of traditional methods, researchers began using deep learning models for geocoding. These models predict geographic locations directly from text, reducing the need for external databases and improving generalization (Yao, 2020; Fornaciari and Hovy, 2019; Huang et al., 2022). Early approaches treated geocoding as either a coordinate prediction task. For example, Liu et al. (Liu and Inkpen, 2015) used deep learning to estimate Twitter user locations from text, achieving good results. Radford et al. (Radford, 2021) developed a model to predict geographic coordinates directly from event text. However, these regression-based methods often struggled with the continuity and infinite nature of geographic coordinates, leading to learning difficulties and performance degradation due to data quality issues.

In response to the challenges faced by regression-based models, researchers explored grid-based classification approaches, where the Earth's surface is divided into discrete grids, and models predict the corresponding grid category based on input addresses (Kulkarni et al., 2020; Viegas, 2021; Gritta et al., 2018; Fornaciari and Hovy, 2019; Cardoso et al., 2019; Serdyukov et al., 2009; DeLozier et al., 2015). For example, Cardoso et al. (Cardoso et al., 2019) combined grid classification with coordinate regression, using context-aware word embeddings and bidirectional LSTM networks to transform text and predict grid categories. This hybrid approach addressed some limitations of pure classification models but still faced challenges related to the high dimensionality of the output space, particularly in fine-grained geocoding tasks.

Recently, researchers have begun addressing this problem using a Seq2Seq formulation (Kothari and Sohoney, 2022; Reddy et al., 2022; Liang et al., 2024), employing an encoder-decoder architecture. The input is an address, and the target is an alphanumeric string generated by hierarchical grid systems like H3, and S2[5]. This approach has proven to be more effective than others.

3 Methodology

We formulated geocoding as a hexagonal grid prediction problem from raw addresses using a Seq2Seq framework. It includes training SentencePiece tokenizer from scratch on a large corpus of data that represents the vocabulary of addresses from different Indian states and fine-tuning of language models to generate alphanumeric H3 cells.

3.1 H3 grid system

We selected the H3 grid system for efficient spatial representation and robust hierarchical indexing. It enhances the model's ability to understand relationships among neighbouring addresses as nearby hexagons share most bits, thereby enforcing embeddings of neighboring addresses to be closely related. A detailed comparison of H3 with other grid systems is provided in the Appendix A.

Encoding H3 Indices for Model Learning: We designed our tokenizer to treat each bit of the H3 index as a separate token, helping the model learn the hierarchical structure of these indices. We achieved this by using unique delimiters, ‘**`^$`**’ and ‘**`$^`**’[6], and incorporating all possible combinations into the tokenizer. This encoding approach enables the model to comprehend every single bit of index effectively.

[5]https://s2geometry.io/

[6]For example, an H3 index of ‘`893db620b13ffff`’ is encoded as: `^$8$^^$9$^^$3$^^$d$^^$b$^^$6$^^$2$^^$0$^ ^$b$^^$1$^^$3$^^$f$^^$f$^^$f$^^$f$^`

3.2 Training Strategy

Training a single model for India is inefficient due to data imbalance, diverse address formats (refer appendix E), leading to bias towards states with more data. To address this, we created individual models for each state. We began with a city-based model for Nagpur, then expanded to other states using Flan-T5-base and Llama-3-8b models.

3.2.1 Training Tokenizer from scratch

We trained a SentencePiece tokenizer (Kudo, 2018) from scratch on 67 million Indian addresses to handle regional variations. This improves the model's ability to accurately parse and process diverse address formats. Refer Appendix D.1 for the detailed comparison between our custom tokenizer and Vanila T5 tokenizer.

The impact of the trained tokenizer is shown in Table 1 which clearly demonstrates that the address is tokenized into coherent units such as *“nagsen”*, *“nagar”*, *“bhim”*, and *“chowk”*, improving model's understanding of the address. We also explored treating pin codes as special tokens to ensure the tokenizer would not split them, but this resulted in drop in accuracy. For more detailed explanation please refer Appendix D.2.

3.2.2 T5 Training

For the T5 model, we fine-tuned the Flan-T5-base[7] variant, which has 220 million parameters, specifically for the task of hierarchical H3-cell prediction. The core of our training involved optimizing the T5 model to predict hierarchical H3-cell accurately. We employed the CrossEntropy loss function (refer appendix F) to enhance the model's predictive capabilities. During evaluation, we used accuracy and haversine distance as metric to assess model performance, reflecting the correctness and geographical relevance of the predictions.

3.2.3 Llama-3 Fine-tuning

For the Llama-3 model, we selected the Llama3-8b variant, which has ∼8 billion parameters. We fine-tuned Llama3-8b model using QLoRA (Dettmers et al., 2023), with detailed information provided in Appendix H.1. We refer to this QLoRA fine-tuned version of Llama3-8b as *QLF-Llama-3* (QLoRA Fine-tuned Llama-3). Our training strategy focused

[7]We initially tested all five variants of the Flan-T5 model to identify the most suitable one for our task. Performance comparison among these is provided in appendix B.

Original Address	Vanila T5 tokenizer	Our tokenizer
XX, nagsen nagar, bhim chowk, jaripatka, nagpur, maharashtra, 440014	[‘_XX’, ‘,’, ‘_, ‘nag’, ‘s’, ‘en’, ‘_, ‘n’, ‘a’, ‘gar’, ‘,’, ‘_’, ‘b’, ‘him’, ‘_’, ‘c’, ‘how’, ‘k’, ‘,’, ‘_ja’, ‘rip’, ‘at’, ‘ka’, ‘,’, ‘_’, ‘nag’, ‘pur’, ‘,’, ‘_ma’, ‘har’, ‘ash’, ‘tra’, ‘,’, ‘_4’, ‘400’, ‘14’]	[‘_XX’, ‘,’, ‘_nagsen’, ‘_nagar’, ‘,’, ‘_bhim’, ‘_chowk’, ‘,’, ‘_’, ‘jaripatka’, ‘,’, ‘_nagpur’, ‘,’, ‘_maharashtra’, ‘,’, ‘_4400’, ‘14’]

Table 1: Impact of training tokenizer on Indian addresses

on optimizing QLF-Llama-3’s ability to predict accurate H3-cell indices.

4 Data Creation

We extracted data for each state from database[8] of a large Ecommerce platform where delivered latitudes and longitudes were stored against the addresses. We then converted co-ordinates to h3 indices after pre-processing. Since our use case was to build a geocoder at the building or house level, we chose resolution 9, which covers radius of roughly 200 meters[9] of each cell. To maintain consistency and ensure representative location data, we selected the most frequently occurring H3 index for each address. This process involves calculating the frequency of each H3 index associated with an address and selecting the index with the highest count. We prefixed each address with a specific prompt to enhance the model’s understanding. The prompts used for each model are provided in appendix C.

4.1 Data Preprocessing Pipeline

To ensure the quality and consistency of the data, our preprocessing pipeline plays a crucial role. For the context of the readers, an e-commerce delivery database typically includes fields such as *AL1* (Address Line 1), *AL2* (Address Line 2), *landmark*, *city*, *state*, and *pin* (Pincode), all provided by the customer. Our preprocessing pipeline addresses the inconsistencies in gathered co-ordinates and address fields using 2 steps: coordinate validation and extensive address cleaning.

4.1.1 Coordinate Validation

Accurate address-to-coordinate mapping is crucial for our model to learn correct geospatial relationships. It ensures delivered coordinates fall within/nearby respective state’s polygon. We used the point-in-polygon method combined with the Haversine formula[10] as given below:

$$\text{distance}(p, P) = \begin{cases} 0 & \text{if } p \in P \\ \min_{q \in \partial P} \text{Haversine}(p, q) & \text{if } p \notin P \end{cases} \tag{1}$$

where p represents the point coordinates, P represents the polygon, and ∂P denotes the boundary of the polygon.

4.1.2 Address Cleaning

Address lines were cleaned through several steps to ensure consistency and quality:

Address Line Cleaning: Address lines were cleaned by removing extra whitespaces, new lines, tabs, and unnecessary characters. This included lowercasing text, trimming punctuation, and eliminating repeating words. We merged ordinal indicators with numbers and removed sequences over six digits to avoid confusion with pincodes. For instance, converting *" 12, Main Road,, "* to *"12 main road"* and *"1 st Avenue"* to *"1st Avenue"*.

Probabilistic Camel Case Splitting: We applied a probabilistic model to split camel case words based on observed frequencies in the dataset. For instance, *"NewDelhi"* is split into *"New Delhi"* if the transition from lowercase to uppercase occurs frequently. The probability of a split is determined by how often each character pair appears, ensuring accurate reflection of common patterns.

Redundant Phrase Reduction: We eliminated duplicate phrases by keeping the first occurrence of the phrase. For example, converting *"JP Nagar, Bangalore, JP Nagar"* to *"JP Nagar, Bangalore"* and *"Near Gandhi Market Gandhi Market"* to *"Near Gandhi Market"*.

Combining Address Components: At last, we finally combined the cleaned components of various address components in following order: *AL1, AL2, landmarks, city, state, pin* into a complete standardized address.

5 Empirical Study

As it is not efficient to conduct extensive experiments on each state simultaneously, we decided to start with a model for a single city and then expand the best set of experiments to various states.

[8] Data statistics for each state are provided in Table 10.
[9] https://h3geo.org/docs/core-library/restable
[10] https://en.wikipedia.org/wiki/Haversine_formula

This approach allowed us to focus our efforts and resources effectively. We picked two relatively more dense Indian cities: *Nagpur*, located in central India, and *Surat*, situated in western India. The statistics for both are provided in Table 10.

5.1 T5 vs QLF-Llama-3

We examined two widely used Language model backbones, Flan-T5-base and Llama-3-8b. T5 being a smaller model, we fine-tuned all 220M parameters whereas for Llama-3 we used Q-LoRA (Dettmers et al., 2024) and only targeted three modules - K(Key), Q(Query), V(Value) such that pre-trained model gets adapted to our task. Detailed configuration is provided in Appendix H.1. Three different evaluation metrics were studied to assess the performance of our models - Mean distance error *(Mean)*, 90th percentile *(P90)*, and 99th percentile *(P99)* distance errors. It is evident from the results in Table 2 that T5 outperformed QLF-Llama-3 across all metrics. We concluded that QLoRA alone is not capable enough to learn such geospatial relationships effectively. It is necessary to either fine-tune the entire model or use ReLoRA (Lialin et al., 2023).

Model	Mean (km)		P90 (km)		P99 (km)	
	N	S	N	S	N	S
T5	**0.6**	**0.4**	**1.2**	**0.8**	**7.2**	**5.6**
QLF-Llama-3	1.3	0.8	2.2	1.9	9.7	7.8

Table 2: Performance comparison between QLF-Llama-3 and T5. "N" denotes Nagpur, and "S" denotes Surat. The best results are highlighted in **bold**.

We investigated the embedding quality of T5 and QLF-Llama-3 by selecting addresses from 10 regions of Surat and visualizing their embeddings with t-SNE (Figure 1). The T5 model's t-SNE plot shows well-defined, tightly packed distant clusters with clear separation. In contrast, QLF-Llama-3's t-SNE plot shows dispersed clusters with noticeable overlap, suggesting it struggles to cluster geospatially close addresses.

5.2 Geographic Expansion using Transfer Learning

In the industry, there is a common need to extend geocoders to different regions. To address this, we explored the possibility of adding new geographies using learning gained from other regions. Our findings revealed that by using the weights of the ex-

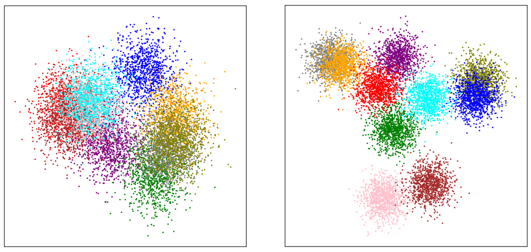

(a) QLF-Llama-3 (b) T5

Figure 1: Embedding representation of addresses taken from 10 different regions of Surat.

isting model as initial weights lead to reduction of convergence time by reducing the number epochs by three times (Figure 2). To achieve this we did two experiments. In the first, we fine-tuned the pre-trained T5 weights. In the second, we initialized weights from an already fine-tuned city model (Nagpur) and then trained the model. Our goal was to see if starting with region-specific weights offered performance improvements and faster convergence. We picked *Bihar*, *Delhi*, and *Gujarat* to assess the performance differences in these two scenarios.

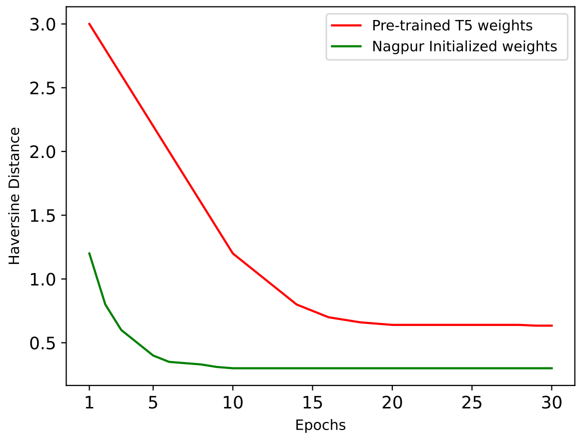


Figure 2: Impact of Transfer Learning on Bihar

To illustrate the impact of transfer learning, we compared the convergence rates and accuracy improvements over training epochs on the evaluation set of Bihar. Figure 2 shows the model trained for 30 epochs with pretrained T5 weights and Nagpur initialized weights. In the first scenario, the distance error decreased from ∼3 km to ∼0.5 km over 30 epochs. In the second scenario, the distance error dropped from ∼1.2 km to ∼0.4 km within 10 epochs. This demonstrates that the model with Nagpur initialized weights converged faster and achieved lower geocoding error in significantly

State	Mean (km)		P90 (km)		P99 (km)	
	PW	NIW	PW	NIW	PW	NIW
Bihar	3.5	**2.1**	7.2	**4.8**	15.3	**7.2**
Delhi	2	**0.5**	4.5	**0.6**	15.1	**8.6**
Gujarat	6.7	**4.1**	8.4	**7.6**	16.5	**9.7**

Table 3: Performance comparison in two scenarios: *PW* refers to generic Pre-trained Flan-T5 weights, and *NIW* refers to Nagpur Initialized Weights. The best results are highlighted in **bold**.

	Seen		Unseen	
	Mean (↓%)	P99 (↓%)	Mean (↓%)	P99 (↓%)
Bihar	*62.5*	*84.7*	*47.3*	*71.8*
Delhi	*30.6*	*71.7*	*22.6*	*57.1*
Gujarat	*27.5*	*84.1*	*22.2*	*73.7*

Table 5: Performance comparison of our approach with Google Maps.

fewer epochs.

Table 3 indicates, initializing model weights with already trained model of another region led to improved performance across all states compared to starting from generic pretrained weights.

5.3 Data Constrained Learning

For smaller states, we have limited data (<0.1M). Training a data-hungry model like Flan-T5 and achieving good performance on a task like geocoding with such limited data is challenging. Therefore, we explored the impact of adjusting batch size and accumulation steps to optimize model's performance under these constraints. Large batch size and accumulation steps can speed up convergence but may miss data variations, while smaller values capture nuances leading to better accuracy but convergence takes more time.

To explore this balance, we conducted experiments across several states with varying dataset sizes. The results are summarized in Table 4, highlighting the impact of different batch sizes and accumulation steps on model performance.

State	Train	Batch size	Acc. steps	Mean (km)	P99 (km)
HP	∼96k	8	4	4.8	8.7
		4	2	4.3	7.8
		2	**1**	**3.8**	**6.9**
Tripura	∼58k	4	2	4.3	9.4
		2	2	4.1	9.1
		2	**1**	**4.1**	**9**
Goa	∼32k	4	2	2.6	6.2
		2	1	2.5	5.9
		1	**1**	**2.4**	**5.6**

Table 4: Impact on performance with different Batch size and Accumulation steps (Acc. steps). The best configuration and its result are highlighted in **bold**.

Results in Table 4 clearly shows that in states with less data, lower batch sizes and accumulation steps significantly enhanced performance. For example, in Himachal Pradesh (HP), using batch size of 2 & accumulation step of 1 led to a mean error of 3.8 km & 99th percentile error of 6.9 km, compared to batch size of 8 & accumulation step of 4, which resulted in mean error of 4.8 km & 99th percentile error of 8.7 km. This approach ensures frequent parameter updates, capturing patterns in limited data effectively. Final configurations for each state are provided in Table 11.

5.4 Industry Benchmark Comparison

We compared the performance of our geocoding pipeline against Google Maps, which is widely regarded as one of the best in class globally. We present the comparison for three states here (in Table 5) where the difference in results is significant, while the detailed results for all states are provided in Table 12 (Appendix I). To ensure the robustness of our approach, we also compared the Google Maps results on H3 cells that were not in the training data. This is shown by the *"Seen"* and *"Unseen"* categories in Table 5. The table displays the percentage reduction in all metrics for both seen and unseen data categories.

Table 5 shows significant performance gap between GeoIndia and Google Maps. For example, in Bihar, GeoIndia achieved a mean distance error reduction of 62.5% for seen data and 47.3% for unseen data. Similar trends in other states show our approach significantly outperforms Google Maps, even in emerging geographies.

6 Real World Deployment

We deployed *GeoIndia* for a leading Indian E-Commerce platform, powering GeoFencing, Hub allocation and Route Optimization for the past 4 months. To ensure a smooth experience with real-time inferencing (<100ms), we optimized model latency from ∼700ms to ∼80ms using Nvidia TensorRT (NVIDIA). See Appendix J for production workflow details.

7 Conclusion & Future Work

In this work, we proposed *GeoIndia*, an effective solution that addressed complexities in geocoding diversely structured addresses. The proposed approach suggests that the region-specific fine-tuning leads to faster convergence. Adjustments to parameters such as batch size and accumulation steps in data-scarce settings were discussed to achieve higher accuracy. We went on to discuss low latency model serving infrastructure with coverage across all states in India and concluded the effectiveness of the proposed solution by comparing it with top industry benchmark (Google Maps).

In future, we will study the impact of noise, such as incorrect pincodes, on geocoding errors. Additionally, we plan to explore complete fine-tuning of different LLMs.

References

Ana Bárbara Cardoso, Bruno Martins, and Jacinto Estima. 2019. Using recurrent neural networks for toponym resolution in text. In *Progress in Artificial Intelligence: 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Vila Real, Portugal, September 3–6, 2019, Proceedings, Part II 19*, pages 769–780. Springer.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Grant DeLozier, Jason Baldridge, and Loretta London. 2015. Gazetteer-independent toponym resolution using geographic word profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Preprint*, arXiv:2305.14314.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Tommaso Fornaciari and Dirk Hovy. 2019. Geolocation with attention-based multitask learning models. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 217–223.

Daniel W Goldberg, John P Wilson, and Craig A Knoblock. 2007. From text to geographic coordinates: the current state of geocoding. *URISA journal*, 19(1):33–46.

Milan Gritta, Mohammad Taher Pilehvar, and Nigel Collier. 2018. Which melbourne? augmenting geocoding with maps. Association for Computational Linguistics.

Jizhou Huang, Haifeng Wang, Yibo Sun, Yunsheng Shi, Zhengjie Huang, An Zhuo, and Shikun Feng. 2022. Ernie-geol: A geography-and-language pre-trained model and its applications in baidu maps. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3029–3039.

Morteza Karimzadeh, Wenyi Huang, Siddhartha Banerjee, Jan Oliver Wallgrün, Frank Hardisty, Scott Pezanowski, Prasenjit Mitra, and Alan M MacEachren. 2013. Geotxt: a web api to leverage place references in text. In *Proceedings of the 7th workshop on geographic information retrieval*, pages 72–73.

Morteza Karimzadeh, Scott Pezanowski, Alan M MacEachren, and Jan O Wallgrün. 2019. Geotxt: A scalable geoparsing system for unstructured text geolocation. *Transactions in GIS*, 23(1):118–136.

Govind Kothari and Saurabh Sohoney. 2022. Learning geolocations for cold-start and hard-to-resolve addresses via deep metric learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 322–331.

T Kudo. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Sayali Kulkarni, Shailee Jain, Mohammad Javad Hosseini, Jason Baldridge, Eugene Ie, and Li Zhang. 2020. Spatial language representation with multi-level geocoding. *arXiv preprint arXiv:2008.09236*.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2023. Relora: High-rank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*.

Linlin Liang, Yuanfei Chang, Yizhuo Quan, and Chengbo Wang. 2024. A hierarchy-aware geocoding model based on cross-attention within the seq2seq framework. *ISPRS International Journal of Geo-Information*, 13(4):135.

Michael D Lieberman and Hanan Samet. 2012. Adaptive context features for toponym resolution in streaming news. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 731–740.

Ji Liu and Diana Inkpen. 2015. Estimating user location in social media with stacked denoising auto-encoders. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 201–210.

Meta. Introducing meta llama 3: The most capable openly available llm to date. https://ai.meta.com/blog/meta-llama-3/.

NVIDIA. Tensorrt. https://developer.nvidia.com/tensorrt/.

Benjamin J Radford. 2021. Regressing location on text for probabilistic geocoding. *arXiv preprint arXiv:2107.00080*.

Yaswanth Reddy, Sumanth Sadu, Abhinav Ganesan, and Jose Mathew. 2022. Address location correction system for q-commerce. In *Proceedings of the Second International Conference on AI-ML Systems*, pages 1–7.

Pavel Serdyukov, Vanessa Murdock, and Roelof Van Zwol. 2009. Placing flickr photos on a map. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 484–491.

Diogo Alexandre Araujo Viegas. 2021. *Toponym Resolution in Text with Neural Language Models*. Ph.D. thesis, Instituto Superior Técnico Lisbon, Portugal.

Xiaobai A Yao. 2020. Georeferencing and geocoding.

Wei Zhang and Judith Gelernter. 2014. Geocoding location expressions in twitter messages: A preference learning method. *Journal of Spatial Information Science*, (9):37–70.

A H3 vs Other grid systems

A.1 S2

S2 and H3 are open-source grid systems using 64-bit cell indexes for efficiency in big data. S2 uses square cells, while H3 uses hexagonal, affecting neighbors, sub division, and visualization.

Neighbors: Squares have two types of neighbors: edge-sharing and point-sharing, complicating real-world movement analysis since movements rarely align with the grid. Analysts must consider both neighbor types. Hexagons have only edge-sharing neighbors, simplifying convolutions and data smoothing as only grid distance matters, not geographic distance.

Subdivision: S2 uses an aperture 4 system, dividing each cell into 4 child cells, ensuring that a point indexed to a cell remains within its parent cell's bounds. In contrast, H3 approximates this process since hexagons don't subdivide exactly into 7 child hexagons.

Visualization: Figure 3 illustrates the projection of S2 and H3 cells on the globe. S2 cells (3a), which are square in the system's projection, can appear distorted when visualized on a globe, often looking like quadrilaterals. In contrast, H3 cells, while also subject to map projection distortions, tend to appear less distorted due to their hexagonal shape.

[11]https://opensource.googleblog.com/2017/12/announcing-s2-library-geometry-on-sphere.html

[12]https://observablehq.com/@claude-ducharme/h3-map

Flan-T5 variant	Mean		P90		P99	
	N	S	N	S	N	S
Small	2.3	1.8	5.3	4.7	15.4	10.3
Base	**0.6**	**0.4**	**1.2**	**0.8**	**7.2**	**5.6**
Large	1	0.9	2.9	2.9	8.3	9.6
3B	2.7	2.2	7	5.6	9.2	8.7
11B	4.4	3.6	11.1	8.5	17.4	10.5

Table 6: Performance comparison among the T5 variants. Here N denotes Nagpur and S denotes Surat. All metrics are measured in kilometers. The best result are highlighted in bold.

A.2 Geohash

Geohash encodes locations using a string of characters, forming a hierarchical square grid system known as a quadtree.

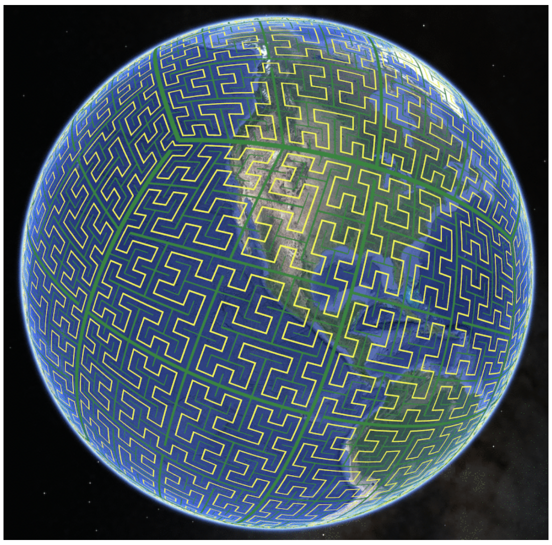
Area distortion: Geohash, which encodes latitude and longitude pairs, results in significant area differences across latitudes. Near the poles, a degree of longitude spans a much shorter distance compared to the same degree near the equator.

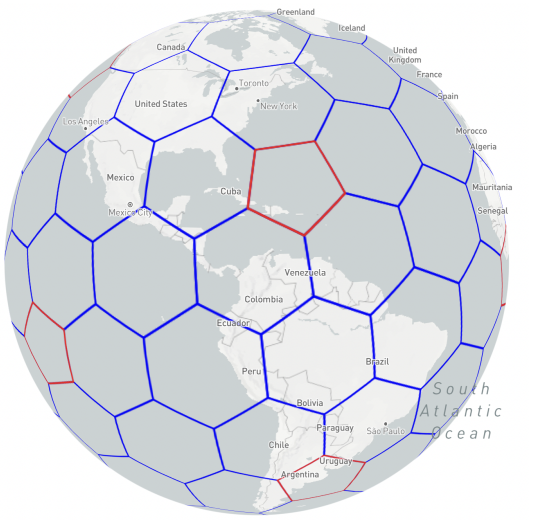
Identifiers: Geohash uses strings for its cell indexes, allowing for arbitrarily precise cells. In contrast, H3 uses 64-bit integers for its cell indexes, which can be converted to strings if necessary. The integer representation offers higher performance due to faster operations compared to strings. However, since the indexes are of fixed size, H3 has a maximum resolution it can encode.

B Performance comparison: T5 Variants

We began by experimenting with all the variants of the Flan-T5 model: Flan-T5-small (60 million parameters), Flan-T5-base (220 million parameters), Flan-T5-large (770 million parameters), Flan-T5-3B (3 billion parameters), and Flan-T5-11B (11 billion parameters). Our experiments were conducted on data from Nagpur and Surat, focusing on the sequence generation task as defined in this paper. The results for each model are summarized in Table 6.

It can be clearly seen in Table 6 that the Flan-T5-base variant consistently outperformed the others


(a) S2 Projection (Image credit[11])



(b) H3 Projection (Image credit[12])

Figure 3: Projections of S2 and H3 on the globe.

in all the metrics. The Flan-T5-base model, with 220 million parameters, strikes a balance between model complexity and the ability to generalize well. For instance, in Nagpur, the Flan-T5-base achieved a mean error of 0.6 km and a P99 error of 7.2 km, whereas the Flan-T5-11B, despite its larger size, had a mean error of 4.4 km and a P99 error of 17.4 km. The smaller Flan-T5-small had a mean error of 2.3 km and a P99 error of 15.4 km, showing that it lacked the capacity to capture the intricate details necessary for precise geocoding. The larger models like Flan-T5-3B and Flan-T5-11B might suffer from overfitting, especially given the variability and complexity of Indian addresses.

C Prompts

We added a specific prompt before each address to help the model understand better. Table 7 shows the prompts used for each model.

D Tokenizer Experimentations

D.1 Performance comparison: Custom vs Vanila T5 Tokenizer

As previously mentioned in the paper, we utilized a custom-trained tokenizer tailored to our use case. We tested performance using both the vanilla T5 tokenizer and the custom-trained tokenizer over two cities, Nagpur and Surat. The results of this comparison are presented in Table 8.

D.2 Treating Pincode as special tokens

We initially explored treating the entire pin code as a single token (*"440014"* as compared to *"4400"* and *"14"*) during our experiments. However, this approach led to suboptimal results. Specifically, it increased the mean distance error from 0.6 km to 2.6 km in Nagpur. By splitting the pin code into hierarchical segments (e.g., *"4400"* representing a larger region and *"14"* representing a smaller locality), the model achieved better performance.

This segmentation leverages the inherent hierarchical structure of the Indian Postal Index Number (PIN) system, where each digit encodes progressively smaller geographical regions. This allowed the model to learn geographical relationships more effectively and significantly reduced the overall vocabulary size, which in turn saved computational resources. As a result, we have retained the segmented approach. For additional reference, the structure of Indian PIN codes is detailed on Wikipedia[13].

E Performance Comparison: Pan-India vs State-wise models

Before transitioning to state-specific models, we initially trained a single model for all Indian states, which we referred to as the Pan-India model. We then compared its performance with that of individual state-based models. The results are presented in Table 9.

[13]https://en.wikipedia.org/wiki/Postal_Index_Number

Model	Prompt
T5	Find the H3-index (15 bits alphanumeric representation of latitude and longitude) corresponding to the address: *{address}*
QLF-Llama-3	Instruction: Find the H3-index (15 bits alphanumeric representation of latitude and longitude) corresponding to the address: *{address}* Response: *{H3-index}*

Table 7: Prompts

City	Mean (km)		P90 (km)		P99 (km)	
	Custom	Vt5	Custom	Vt5	Custom	Vt5
Nagpur	**0.6**	3.8	**1.2**	8.1	**7.2**	19.3
Surat	**0.4**	2.5	**0.8**	5.8	**5.6**	12.8

Table 8: Performance comparison between custom and Vanila T5 (Vt5) tokenizer.

State	Model	Mean (km)	P90 (km)	P99 (km)
Delhi	state	0.5	0.6	8.6
	pan-india	0.7	1.0	10.7
Himachal Pradesh	state	3.8	4.7	6.9
	pan-india	5.9	10.9	17.7
Haryana	state	2.0	3.8	26.9
	pan-india	2.9	7.0	28.6

Table 9: State-based vs pan-india model performance comparison

F Training Objective

In this section, we provide a detailed explanation of the training objective and loss function used in our model. The model follows a sequence-to-sequence approach, where each token in the input sequence is processed to generate a corresponding output token at each time step. The loss is computed for each individual output token in the sequence, and the overall objective is to minimize this loss across all tokens. For example, consider an input address: *"XX, nagsen nagar, bhim chowk, jaripatka, nagpur, maharashtra, 440014"* and the corresponding output tokens: `^$8$^^$9$^^$3$^^$d$^^$b$^^$6$^^$2$^^$0$^ ^$b$^^$1$^^$3$^^$f$^^$f$^^$f$^^$f$^`

After tokenization, the input sequence is represented as:

[25602, 108, 103, 1844, ...]

and the output sequence is:

[32108, 32109, 32106, 32100, ...]

At each time step, the model generates a probability distribution over the entire vocabulary. For example, at time step 1, the predicted distribution is:

[0.2 , 0.05 , 0.01, ..., 0.65, ...]

Let's assume the predicted probability of target token 32108 is *0.65*, the loss at this step would be calculated as:

$$L_1 = -\log(0.65) \approx 0.43.$$

Similarly, at time step 2, predicted probability distribution is:

[0.05, 0.1, 0.15, ..., 0.7, ...]

So, the predicted probability of target token 32109 is *0.7*, the loss at this step would be calculated as::

$$L_2 = -\log(0.70) \approx 0.36.$$

The total loss for the sequence is then the sum of the individual token losses across all time steps:

$$L_{total} = \sum_i L_i.$$

In this example, the total loss would be approximately:

$$L_{total} = 0.43 + 0.36 + \dots$$

with each token's contribution aggregated to guide the model's training process.

G Data statistics

Table 10 displays the volumes of the training, evaluation, and test datasets used for model creation and testing.

State/City	#Train	#Test	#Eval
Uttar Pradesh	6774636	715238	329704
Maharashtra	4124983	422402	164567
Karnataka	3225645	341366	128323
Bihar	2973674	305415	153213
Andhra Pradesh	2778415	299606	111282
Tamil Nadu	2677789	287302	112636
Rajasthan	2513467	263968	96809
Gujarat	2169534	227639	85836
Delhi	1999972	195807	83193
Jharkhand	1456786	157714	57254
West Bengal	1393252	149801	95143
Haryana	1344594	136493	53721
Telangana	857638	79222	40297
Uttarakhand	771748	81046	31181
Madhya Pradesh	764635	89595	48621
Punjab	550542	56649	37880
Assam	462330	56996	44162
Odisha	453252	58534	40629
Kerala	443570	56498	64606
Chhattisgarh	302431	40495	25297
Himachal Pradesh	96076	13281	8989
Tripura	58577	7199	6018
Meghalaya	37715	4888	2826
Manipur	35606	3410	3212
Goa	32020	3568	3400
Arunachal Pradesh	26569	2915	1957
Nagaland	21168	2100	1475
Mizoram	14053	1412	887
Sikkim	12207	1342	678
Nagpur	250417	10400	25583
Surat	294488	12228	28858

Table 10: Final Data statistics

State	Batch size	Acc. steps
Uttar Pradesh	32	32
Maharashtra	32	16
Karnataka	16	8
Bihar	16	8
Andhra Pradesh	16	8
Tamil Nadu	16	8
Rajasthan	16	8
Gujarat	16	8
Delhi	16	8
Jharkhand	8	8
West Bengal	8	8
Haryana	8	8
Telangana	8	8
Uttarakhand	8	8
Madhya Pradesh	8	8
Punjab	8	4
Assam	4	4
Odisha	4	4
Kerala	4	4
Chhattisgarh	4	4
Himachal Pradesh	2	1
Tripura	2	1
Meghalaya	1	1
Manipur	1	1
Goa	1	1
Arunachal Pradesh	1	1
Nagaland	1	1
Mizoram	1	1
Sikkim	1	1

Table 11: Final batch size and accumulation steps

H Training configuration

The training of our geocoding models was conducted on a high-performance computing instance to ensure efficient processing and optimal performance. We used an instance type of g2-standard-96, which is equipped with 96 vCPUs, 384 GB of memory, and 8 NVIDIA Tesla V100 GPUs.

For the training, we used PyTorch along with the Hugging Face Transformers[14] library to leverage state-of-the-art natural language processing capabilities. We optimized the models using the AdamW optimizer with a learning rate of 3e-4 and a weight decay of 0.01 to prevent overfitting. The models were trained for 10 epochs in the weight-initialized (NIW) scenario and 30 epochs in the pre-trained scenario (PW), ensuring thorough fine-tuning and convergence.

Each state was experimented with multiple sets of batch sizes and accumulation steps. The final set of configuration used during training is provided in the Table 11.

H.1 Llama Training: LoRA configuration

For the Llama-3 model training, we utilized Low-Rank Adaptation (LoRA) to enhance the model's performance. The configuration details for LoRA are as follows:

1. **LoRA attention dimension (r):** *256*, which determines the rank of the low-rank adaptation

[14] `https://huggingface.co/docs/transformers/v4.31.0/en/index`

matrices.

2. **Alpha for LoRA scaling**: *512*, which is a scaling factor for the low-rank matrices.

3. **Dropout probability for LoRA:** *0.1*, which helps in regularizing the adaptation process by preventing overfitting.

4. **Inference mode:** Set to *False*, indicating that the model is in training mode.

5. **Bias:** *None*, meaning no additional bias parameters were introduced in the adaptation.

6. **Task type:** Causal Language Modeling (*CAUSAL_LM*), tailored for the generative nature of the geocoding task.

I Comparison with Google Maps

GeoIndia was able to perform extremely well even in underdeveloped states of India where address complexity is highly severe. The comparison is detailed in Table 12, which shows the percentage reduction in distance error metrics between our approach and Google Maps.

In Uttar Pradesh, the mean distance error saw a reduction of 38.5% in seen data and 27% in unseen data. The 99th percentile error reduced by 85.6% in seen data and 75.5% in unseen data.

In Andhra Pradesh, the mean distance error decreased by 53.7% in seen data and 40% in unseen data, with the 99th percentile error reducing by 88.2% in seen data and 64.3% in unseen data.

Even in states with limited data availability, such as those in North-East India, our approach demonstrated strong performance. For instance, in Arunachal Pradesh, the 99th percentile error reduced by 72.0% in seen data and 52.8% in unseen data, showcasing the robustness and generalization capability of our model. This consistent performance across various metrics highlights the effectiveness of GeoIndia in addressing the unique challenges of Indian address geocoding compared to Google Maps.

J Production Workflow

Our production service is powered by Kubernetes Clusters where we have hosted our models using Nvidia Triton Inference Server. Additionally we also optimized our models with Nvidia-TensorRT, resulting in bringing down model latency from ∼700 ms to ∼80 ms. For model routing we have trained a gating network(kind of a neural network) and also model guardrails have been implemented to ensure production safety. To save the compute on repeated address we have utilized redis for caching with moderate TTL. In the end we have implemented a feedback loop mechanism to iteratively improve our models. The final workflow of our deployed geocoding system is shown in Figure 4.

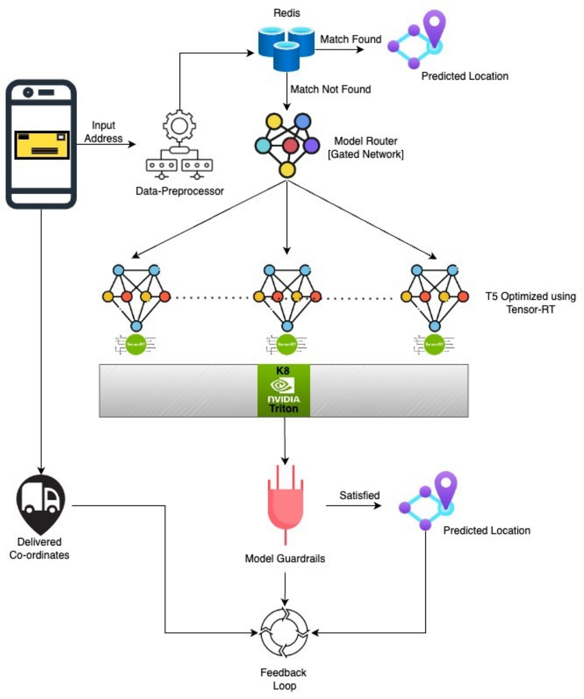


Figure 4: Realtime Geocoding System

State/City	Seen					Unseen				
	Mean (↓%)	Median (↓%)	P75 (↓%)	P90 (↓%)	P99 (↓%)	Mean (↓%)	Median (↓%)	P75 (↓%)	P90 (↓%)	P99 (↓%)
Jharkhand	39.5	25	53.6	55.4	85.2	30.1	18	40.2	42.3	67.7
West Bengal	54.2	25	54.3	65.5	86.5	40	19.8	36.8	43.2	69.6
Uttar Pradesh	38.5	40	51.7	67.4	85.6	27	30.7	39.2	45.1	75.5
Odisha	39.1	38.5	24.7	40.3	85.6	34.3	34.5	17.1	28.4	58.4
Nagpur	38.5	0	42.9	35.5	69.3	25.2	0	33.7	25	60.3
Delhi	30.6	57.1	60.3	34.1	71.7	22.6	49.3	40	26.2	57.1
Surat	22.2	33.3	54.5	31.8	69.2	18.5	23.5	42.8	24.1	62
Andhra Pradesh	53.7	50	52.7	63.1	88.2	40	37.3	39.3	41.9	64.3
Arunachal Pradesh	84.9	45	59.7	69.7	72	68.8	35.6	43.3	61.7	52.8
Assam	82.5	27.8	24.1	44.1	79.1	60.4	23.6	20.9	36.2	64
Bihar	62.5	40	70.8	71.6	84.7	47.3	31.6	46.2	62.6	71.8
Chhattisgarh	38.7	5.6	41.9	55.4	77.9	33.3	4.1	35.1	43.4	65.3
Goa	12.9	11.1	21.4	15.2	72.1	11.1	9.7	15.7	13.2	61.3
Gujarat	27.5	16.7	59.6	29.6	84.1	22.2	11.6	52.2	20.4	73.7
Himachal Pradesh	19.3	15.4	38.5	34.5	87.4	12.7	12.1	25.2	25.8	62.5
Karnataka	41.9	0	12.9	15.9	86.9	37.3	0	11.2	13.1	75.6
Kerela	9.6	16.7	12.2	23.1	81.3	7.9	13.3	10.8	16.8	61.7
Madhya Pradesh	28.3	42.9	43.4	34.1	74.4	23.6	33	38	25.8	49.7
Maharashtra	17.6	33.3	55.3	44	88.5	14.4	24.4	47.9	31.7	73.2
Manipur	6.3	14.3	60.5	50.6	81	5.4	10.8	54.1	35.1	72.8
Meghalaya	15.2	31.6	50.5	43.9	51	11.1	26.3	36.2	28.9	42.7
Mizoram	20.7	0	45.6	50.8	75.1	15.6	0	34.5	42.9	49.7
Nagaland	8.8	36.4	25.2	14.8	83.7	6.9	25.1	18.3	12.2	58.3
Rajasthan	31.3	45.5	41.6	55.4	82.4	25.1	30.9	31.4	37.6	59.9
Sikkim	13.9	9.1	31.9	31.6	68.4	10	7	21.1	23.4	54.3
Tamil Nadu	32	46.2	58.5	68.7	88.2	23.9	41.4	48.2	50.5	73
Telangana	25.6	27.3	41	37.9	84.6	19.1	21.6	35.5	31.3	64.1
Tripura	27.8	27.8	52.5	51.1	78.6	20.9	21	38.7	37.7	62.3
Uttarakhand	63	55.6	5.1	17.7	87.2	42.6	44.9	4	12	60.9
Harayana	26.5	50	63.2	37.2	77.3	19.2	43.7	42.4	26.8	50.3
Punjab	18	0	13.1	14.6	72.4	15.5	0	11.3	10.9	62.9

Table 12: Geocoding metrics relative to Google Maps.