Lightweight Models on Nutribench

By: **Erik Feng, David Bazan, Liam Yaroschuk, David Chang, Kiran Duriseti** |
June 13, 2025

1. Introduction

We investigate whether small-footprint neural architectures can match or exceed the performance of large language models (LLMs) on Nutribench, a benchmark that maps free-form meal descriptions to carbohydrate counts. After establishing naïve baselines (mean regressor, L1/L2-regularised linear models), we develop two lightweight neural systems: (i) an LSTM-based RNN containing less than 1 million parameters, and (ii) a fine-tuned encoder-only transformer with less than 100 million parameters. A systematic Optuna-driven hyper-parameter search lowers validation mean-absolute-error (MAE) from 21 grams (mean predictor) to 7.56 grams with the RNN and 4.44 grams with DistilBERT, surpassing previously reported LLM results while requiring orders-of-magnitude fewer parameters and training time. Exploratory data analysis reveals vocabulary overlap across splits but exposes bimodality and longer query tails in the test set, partially explaining residual generalization gaps. Our findings suggest that, for numeric reasoning tasks grounded in domain text, compact models combined with task-specific tuning can deliver high accuracy without the computational overhead of billion-parameter LLMs.

1.1 Background

Given the rise of cheap, high-calorie processed foods, obesity levels have spiked in areas with easy access to stores with these kinds of foods. These areas are also typically technologically advanced, as the advancements in technology coincide with the prevalence of highly processed foods.

Therefore, there exists a unique opportunity to give rich information about what someone is about to eat based on empirical descriptions.

1.2 Dataset

Therefore, we choose the Nutribench dataset for the simplicity of the task. Given a textual description of the meal, we predict the total carb count (in grams) of the meal. To do this, we develop a model.

Formally our task is as follows:

$$\text{With } S = \{\text{every possible string of words}\}, \text{ and a loss function } L : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$$

$$\text{Given a dataset } d \text{ ɜ } (d_{\cdot,0})^T = S^n, (d_{\cdot,1})^T = \mathbb{R}^{+n}, \text{ we aim to develop a model:}$$

$$m : S \to \mathbb{R}^+ \text{ ɜ } m = \text{argmin}_{\text{m}} \sum_{s \in S} L(m(s), c_s), \qquad c_s \in \mathbb{R}^+ \text{ is the ground truth}$$

For the simplicity, we consider $0 \in \mathbb{R}^+$. Additionally, to follow the conventions of Nutribench, we use the following loss functions:

$$L_\delta(\hat{c}, c_s) = \text{MAE}(\hat{c}, c_s) = |c_s - \hat{c}|$$

$$L_{\mathcal{R}}(\hat{c}, c_s) = \frac{L_\delta(\hat{c}, c_s)}{c_s}$$

$$L_\beta(\hat{c}, c_s) = \text{Acc@7.5}(\hat{c}, c_s) = \mathbb{1}_{|c_s - \hat{c}| \leq 7.5} = \begin{cases} 1 & |c_s - \hat{c}| \leq 7.5 \\ 0 & \text{otherwise} \end{cases}$$

We will also denote $\tilde{L}_x(m_i)$ to denote the $\tilde{L}_x$ loss of model m_i on the validation set.

$$\forall x \ \tilde{L}_x : S \to U \qquad (U \subseteq \mathbb{R}^+)$$

$$\tilde{L}_\delta : (S \to \mathbb{R}^+) \to \mathbb{R}^+, \qquad \tilde{L}_\beta : (S \to \mathbb{R}^+) \to [0, 1], \qquad \tilde{L}_{\mathcal{R}} : (S \to \mathbb{R}^+) \to \mathbb{R}^+$$

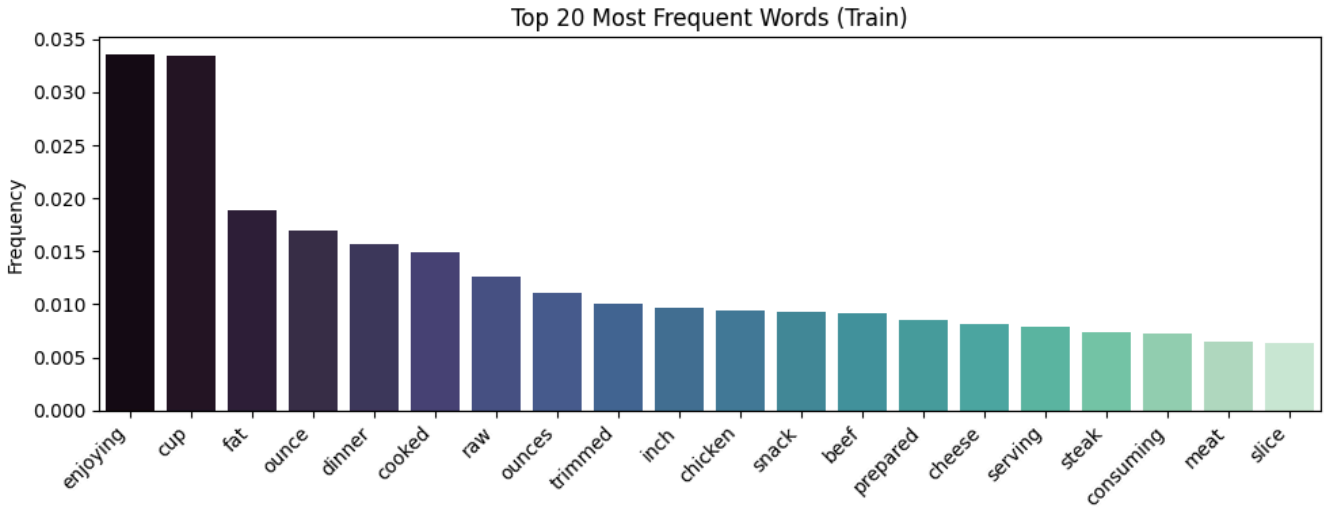
The current highest-performing models on the Nutribench dataset use Large-Language-Models (LLMs) using upwards of 8 billion parameters, combined with prompting techniques like Chain-of-Thought and Retrieval Augmented Generation (RAG). While these models perform well and have the ability to generate even better predictions with more advanced techniques such as tool-calling, they also present a variety of drawbacks.

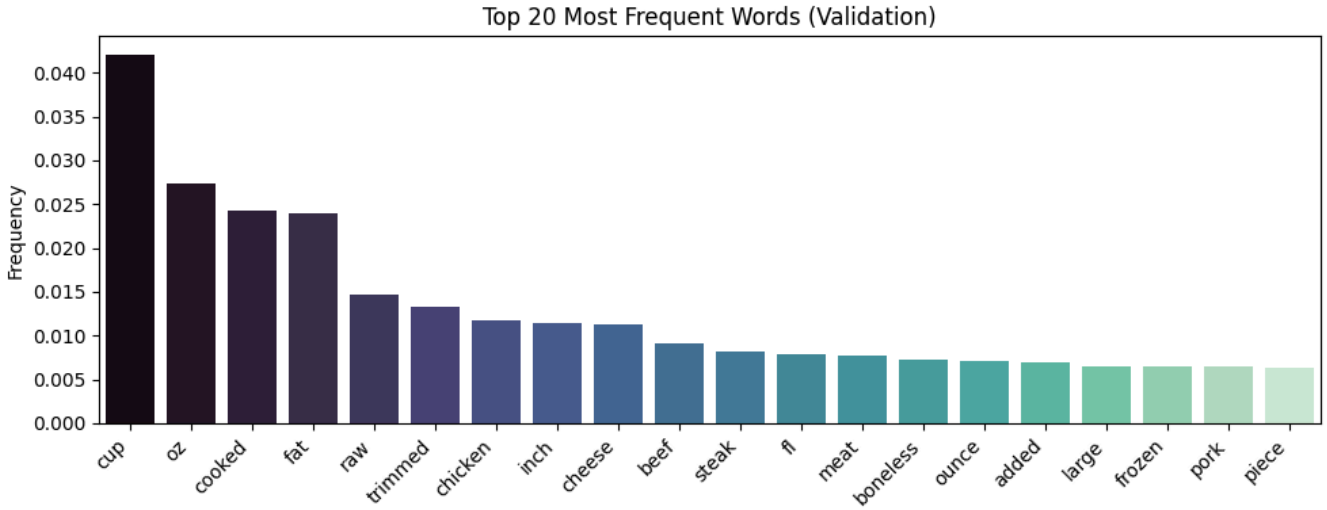
First, as LLMs are non-deterministic, they may not give a proper answers, whether in the right form, the right metric, or providing an answer at all. Additionally, LLMs have questionable scaling properties for this task, in terms of both model and data size, they are hard to run due to how large they are, and can be overkill for this task, as their size likely indicates that there are unused parameters.

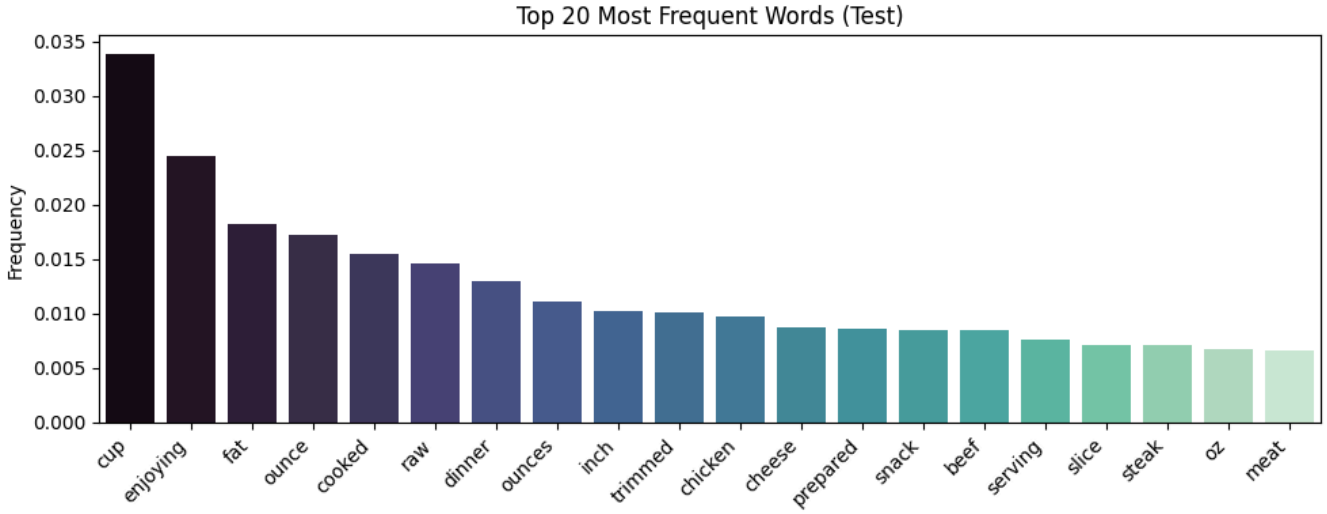
Therefore, we propose 2 models, of which both perform better than the state-of-the-art (SOTA) performance demonstrated by LLMs—an extremely lightweight LSTM model with less than a million parameters, and a fine-tuned encoder-transformer model with less than 100 million parameters.

1.3 Exploratory Analysis of the Data

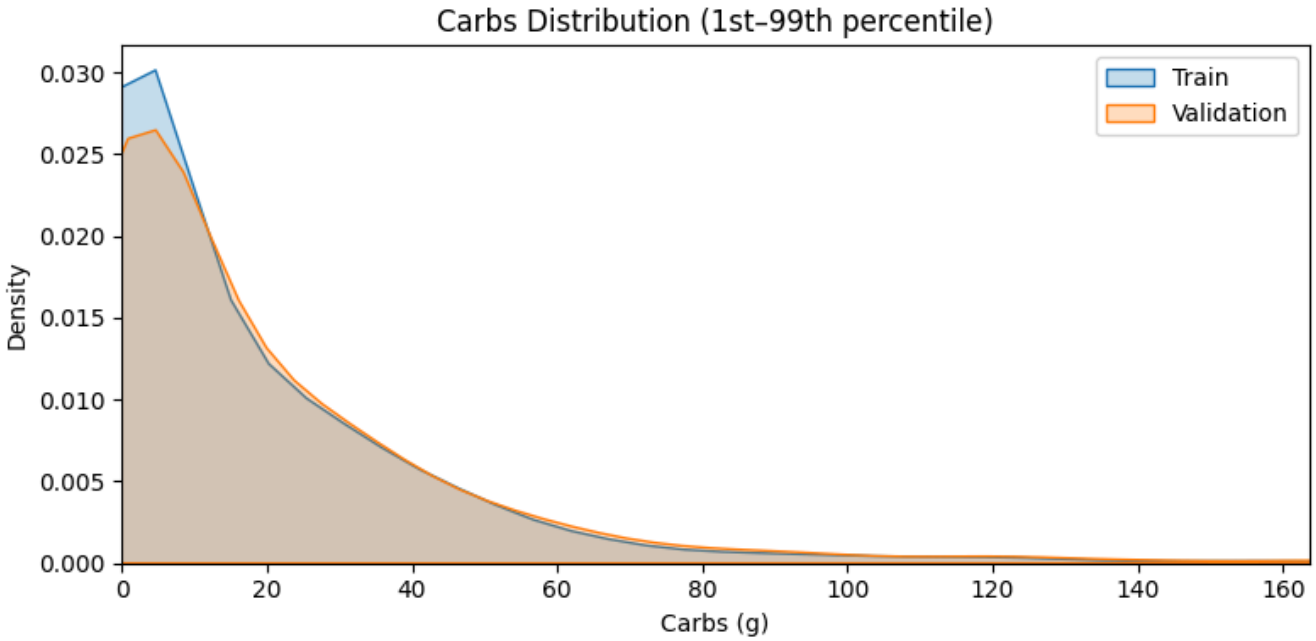
In order to gain some understanding of the data characteristics, we did some exploratory analysis.



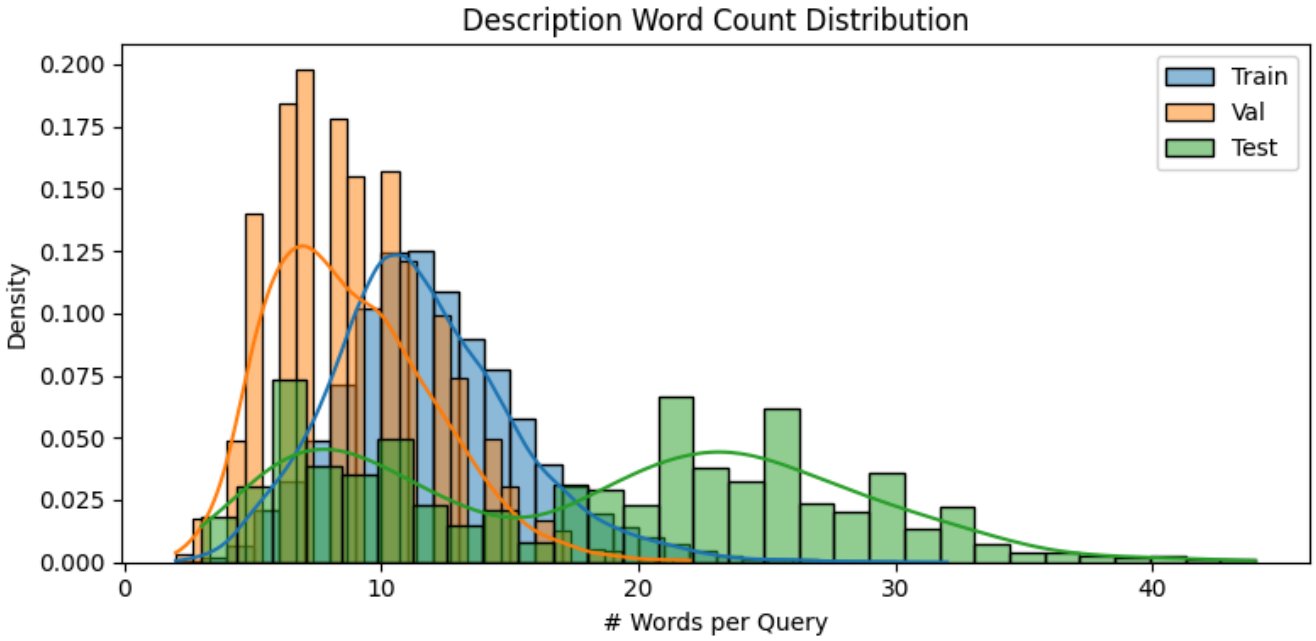




After filtering by stopwords, we can see that the top words in all dataset split are very similar, with quantifiers and common foods making up most of the distribution.



Both the train and validation distribution of carbs are similarly distributed, though train has slightly smaller tails.



All 3 datasets are also similarly distributed, though the test data notably is bimodal and has significantly more longer queries. This may contribute to some unexpected behavior at test-time inference for the models, as expanded in sections below.

2. Models

We first establish a couple of baseline models in order to analyze the performance gains of our models compared to trivial or nearly trivial models.

Denote our training and validation data d_t and d_v, respectively.

2.1 Mean Prediction

2.1.1 Motivation and Architecture

Our first model predicts the mean carbs universally, regardless of the meal description. This crude, yet simple model provides our first benchmark.

$$m_1(s) = \frac{1}{|S_t|} \sum_{s_t \in S_t} s_t$$

Note that since we do not see any s in m_1, the data given to the model does not change its prediction. In fact, since $|S_t|$ and $\sum_{s_t \in S_t} s_t$ are pre-computed, this model runs instantly and only needs to store a single value.

2.1.2 Performance and Discussion

Predictably, this model performs very poorly.

$$\tilde{L}_\delta(m_1) = 21.01$$

$$\tilde{L}_\beta(m_1) = 0.033$$

2.2 Linear Regression

Our second model predicts the carb values using linear regression. Although this model is also simple, it will offer better results just taking the mean carb value. We will use both L1 (Lasso) and L2 (Ridge) regularization in our linear regression model and pick the model with better performance. The chosen model will serve as a non-trivial baseline that we can compare to.

For our Linear Regression model, we will tokenize our data and feed it to the models. The following are the defined models:

$$m_2^{(1)} = \text{argmin}_\text{m} \sum_{s \in S} L_1(m), \qquad m_2^{(2)} = \text{argmin}_\text{m} \sum_{s \in S} L_2(m),$$

$$L_1(m(s), c_s) = L_\delta(m(s), c_s), \qquad L_2(m(s), c_s) = (m(s) - c_s)^2$$

Where our model follows the formula:

$$m_2(x) = wx + b$$

2.2.3 Performance and Discussion

We get the following metrics:

$$L_\delta(m_2^{(1)}) = 15.25$$

$$L_\beta(m_2^{(1)}) = 0.045$$

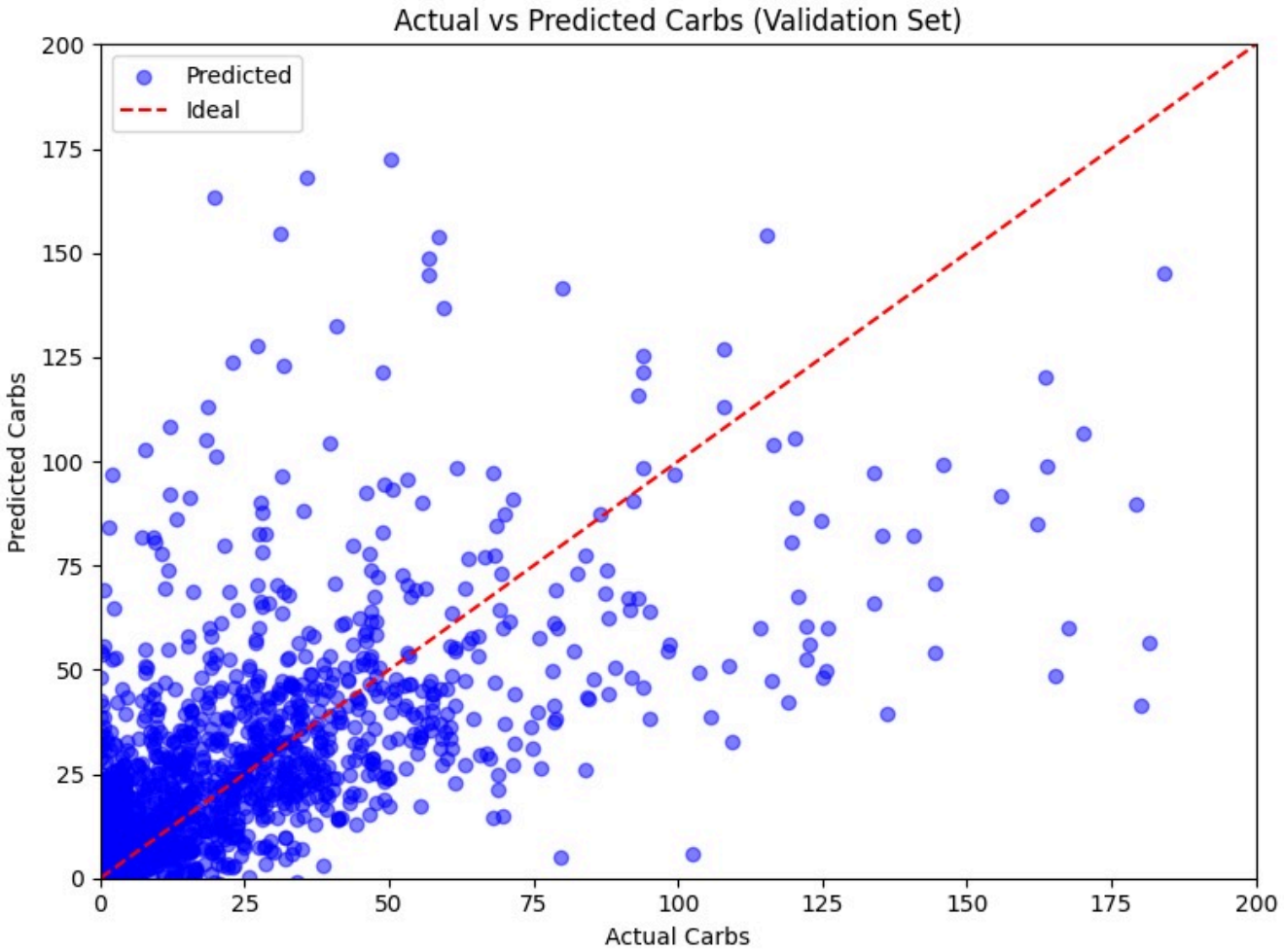
$$L_\delta(m_2^{(2)}) = 14.77$$

$$L_\beta(m_2^{(2)}) = 0.0485$$

The $m_2^{(1)}$ has a validation MAE of 15.25 while $m_2^{(2)}$ has a validation MAE of 14.77.

$m_2^{(2)}$ had a better MAE than the $m_2^{(1)}$ and had a greater accuracy as well. The worst performances of the model came from foods with extremely high carb contents (explored more in the RNN Discussion section).

This behavior is slightly unexpected since we would expect minimizing the L_1 loss would also minimize the L_δ loss, since they are defined to the be the same function.



Our model achieves $r^2 = 0.4421$.

We omitted 13 data points with actual carb values over 200 grams, as the linear regression model appears not to be able to predict over 200 grams.

2.3 RNN

2.3.1 Motivation

Our third model will be using a Recurrent Neural Network in order to predict carb values. This model will serve as an ultra-lightweight alternative for our transformer architecture. Due to the fact that this model is created from scratch, it offers unparalleled computational efficiency in both train and test scenarios.

2.3.2 Architecture

Our model will be made up of two RNN layers alongside dropout regularization, and LSTM. The model also features ADAM optimization and MSE loss. The model will have a 10,000-word vocabulary and include padding after tokenization. The model will be trained for 50 epochs and has a default batch size of 32.

For finding optimal parameters, we used the Optuna library. Our grid search is as follows:

$$H = (d_e, d_r, d_l, \lambda, \alpha), \text{ where}$$

$$d_e \in \{64, 128, 256\}$$

$$d_r \in \{16, 17, \ldots, 128\}$$

$$L \in \{16, 32, 64\}$$

$$D \in [0.2, 0.5]$$

$$\alpha \in [1 \times 10^{-2}, 1 * 10^{-4}]$$

Where d_e is the embedding output dimension, d_r is the first RNN layer's unit count, d_l is the LSTM unit count, λ is the dropout rate, and α is the learning rate.

It is important to note that the second RNN layer's unit count was not calculated independently and instead was set to $\lfloor R/2 \rfloor$.

2.3.3 Difficulties and insights.

During training, we originally only had a single layer RNN. While this gave promising results, the model suffered from severe overfitting. In an attempt to improve the model we decided to add a dropout regularization and stack another RNN layer. With the new dropout regularization, the model no longer overfitted and the extra layer helped to lower MAE. Since the model is very light, there were no issues with training and only took a couple of minutes. The grid search on the model took about an hour and 30 minutes to compete.

2.3.4 Performance

MAE $\tilde{L}_\delta$	**Embedding Dimension** d_e	**RNN Units** d_r	**LSTM Units** d_l	**Dropout Rate** λ	**Learning Rate** α	**Training Duration**
7.56	64	128	32	0.399	0.001	03:03

MAE $\tilde{L}_\delta$	Embedding Dimension d_e	RNN Units d_r	LSTM Units d_l	Dropout Rate λ	Learning Rate α	Training Duration
7.65	256	38	64	0.470	0.0008	02:54
7.68	64	78	16	0.237	0.002	02:19
7.79	64	74	16	0.295	0.002	02:18
7.92	128	125	32	0.413	0.001	03:25

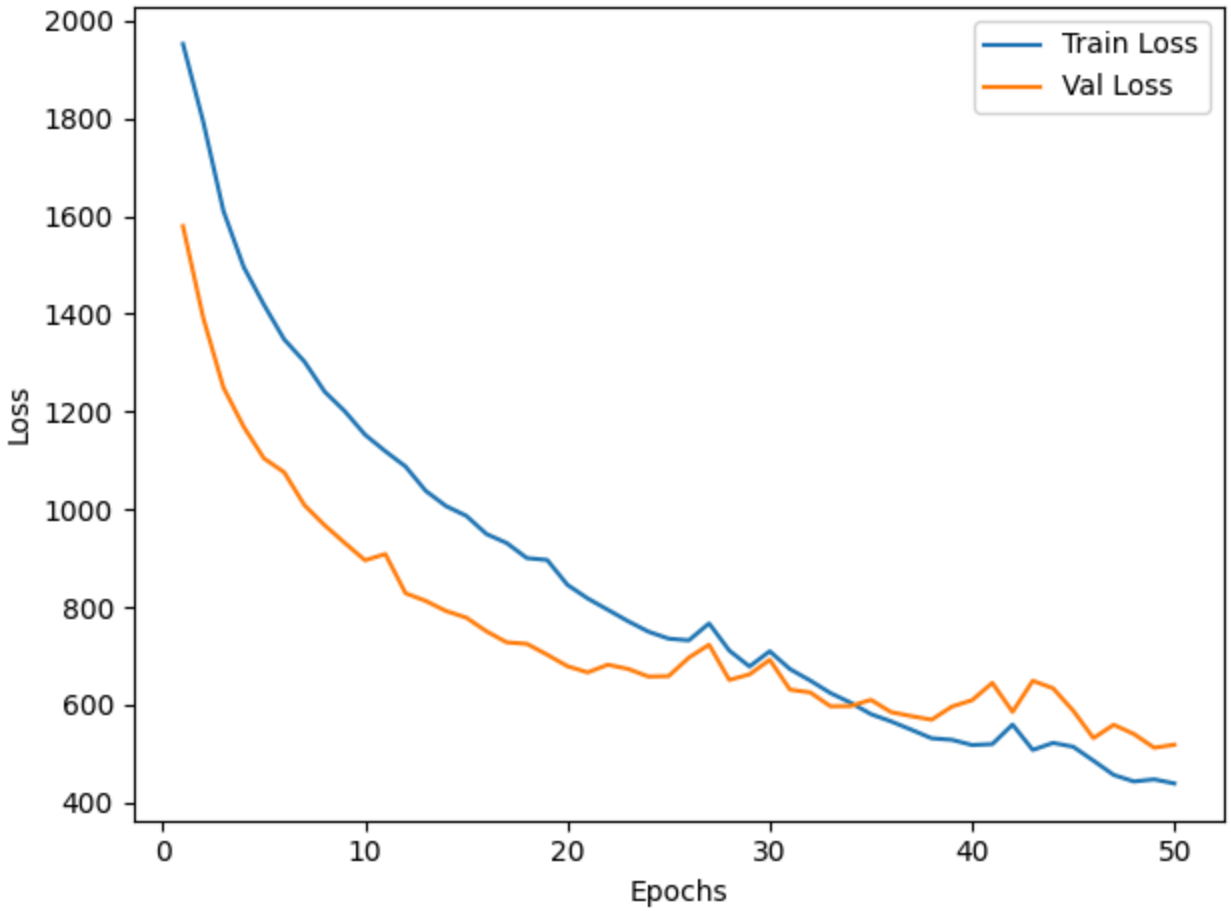
We achieve our best MAE of 7.56 with $d_e = 64$, $d_r = 128$, $d_l = 32$, $\lambda = 0.399$, and $\alpha = 0.0017$. The top 5 models all reach very similar results ($\tilde{L}_\delta < 8$).

Let us denote the top model by m_r^*. We have the following metrics:

$$\tilde{L}_\delta(m_r^*) = 7.56$$

$$\tilde{L}_\beta(m_r^*) = .733$$

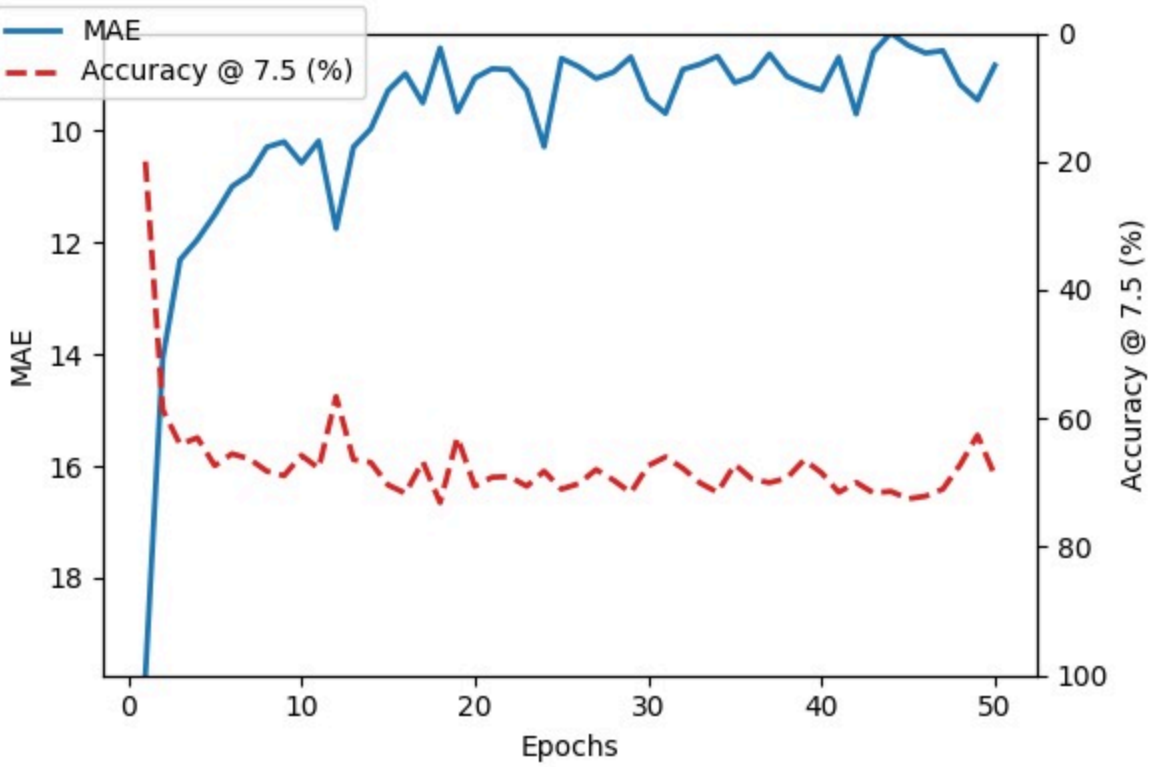
The following is a graph comparing Training Loss vs Validation Loss:



The RNN model successfully learns, shown by the decreasing training and validation losses. The losses also seem to converge, suggesting that increasing the epochs will not improve the

model.

The following is a graph showing MAE and Accuracy over epochs.



The graph shows the model's accuracy and MAE plateaus as epochs increase. This shows that the model is once again learning cannot be improved by increasing the number of epochs.

2.3.5 Discussion

We can see that the top 5 models performed better than SOTA. All models took around 3 minutes to train but one trial failed. The trial that failed stated that it failed with a value of nan. Using the parameters that the trial used shows that MAE increased and gave an L_ß of .175. We are not sure why this trial failed but it is an interesting outcome.

The RNN was able to capture various patterns. One pattern in specific was that it consistently performed well on foods with medium-high carbohydrates content. Some examples are listed below:

Query s	**Carbs** c_s	**Prediction** $\hat{c}$	**Error** L_δ	**Relative Error** $L_\mathcal{R}$
One large burrito filled with beans and cheese	84.01	82.94	1.07	.0127
One cup of MALT-O-MEAL Raisin Bran Cereal.	47.41	41.95	5.47	.1151

Query s	**Carbs** c_s	**Prediction** $\hat{c}$	**Error** L_δ	**Relative Error** $L_\mathcal{R}$
One cup of steamed white rice.	44.72	50.35	5.63	.1258

In these examples, the RNN was able to associate certain keywords like “burrito” and “cereal” with having high carb counts, demonstrating that it can recognize carb-dense foods. It is important to note that the model does not do well when the food is super high in carbs, as shown in the highest errors tables.

Although the model did well with foods with high carbohydrates content, it had trouble with foods that contained zero carbs.

Query s	**Carbs** c_s	**Prediction** $\hat{c}$	**Error** L_δ	**Relative Error** $L_\mathcal{R}$
Half a roasted duck with skin removed.	0.0	1.08	1.08	-
One cubic inch of pork.	0.0	6.85	6.85	-
A regular steak.	0.0	6.66	6.66	-

The model tries to predict these zero carb foods to have carbs.

Query s	**Carbs** c_s	**Prediction** $\hat{c}$	**Error** L_δ	**Relative Error** $L_\mathcal{R}$
“One whole strawberry pie.”	707.52	209.89	497.63	0.7033
“One complete recipe of praline candies.”	540.48	93.33	447.15	0.8273
“A 2-layer white cake made from a recipe without frosting.”	507.48	74.69	432.67	0.8528

Looking at the highest error queries, we also found similar results. The query with the highest error is “One whole strawberry pie”. It is unclear how big this pie might be, so the model gives a possibly reasonable prediction of 209 grams of carbs while the ground truth is 707 grams of carbs. Overall, it seems like the model performs badly on vague and unclear queries.

Query s	**Carbs** c_s	**Prediction** $\hat{c}$	**Error** L_δ	**Relative Error** $L_\mathcal{R}$
“One inch of gluten-free thick crust pizza with meat.”	1.90	52.04	50.13	26.32

Query s	**Carbs** c_s	**Prediction** $\hat{c}$	**Error** L_δ	**Relative Error** $L_\mathcal{R}$
“French fries with chilli"	2.52	53.60	51.08	20.22
“One multigrain restricted potato chip”	1.30	23.41	22.10	16.91

Looking at the highest relative error queries, we found some interesting results. Queries that are vague and misleading have the highest relative errors. One specific query is the “French Fries with chilli” that has a ground truth of 2.52 grams of carbs. It is unclear if this means an individual fry or a couple of fries, which leads the model to predict 53.6 carbs. Another query that confuses the model is “One multigrain restricted potato chip”. It is unclear if this refers to one singular chip or an entire bag and so the model predicts the carb content of a whole bag.

2.4 Encoder Transformer

2.4.1 Motivation for the Transformer

Furthering the RNN architecture, we use transformers, specifically the encoder architecture. However, due various training obstacles such as time and hardware constraints, we are not able to train a transformer from scratch. Therefore, we employ a fine-tuning technique to utilize pre-trained transformers so that we may get the performance benefits of large transformer architectures while still being able to train the transformer on our dataset.

2.4.2 Architecture and Hyperparameters

For our transformer, we must use a tokenizer in order to feed our transformer our data. As seen below in our choice of hyperparameters, we simply use the fine-tuned model's tokenizer. Next, our tokenized data is fed directly into the transformer. We train with an Adam optimizer.

At first we explored a variety of Transformer models, focusing on BERT. Due to a lucky initial hyperparameter choice, we found that our training process was yielding SOTA-beating performance on the validation set within 10 epochs. Because we were using BERT-Base, we switched to BERT-large, believing the tripled parameter count would result in better performance. However, this was found not to be true not only for that case, but for a general case. As we will show, an early exhaustive grid search yielded poor results for the larger BERT models. This is likely due to overfitting and/or having too many parameters resulting lots of vanishing gradients across such a model (even with the transformer architecture).

While our initial hyperparameter search was fully exhaustive, our second grid search used methods such as early stopping to optimize the speed at which we explored the grid. As before, the Optuna library was used to do grid searching.

For hyperparameters, our search grid is as follows:

$$H = (\gamma, \lambda, \alpha, \text{epochs}{=}50)$$

$$\gamma \in \{\text{bert-base-cased}, \text{bert-base-uncased}, \text{roberta-base}, \text{distilbert-base-uncased}, \text{distilbert-base-cased}\},$$

$$\lambda \in \{0, 0.005, 0.01\},$$

$$\alpha \in \{1 \times 10^{-3}, 5 \times 10^{-4}, 1 \times 10^{-4}, 5 \times 10^{-5}, 1 \times 10^{-5}, 5 \times 10^{-6}\}.$$

where γ is the model, λ is the regularization constant (weight-decay), and α is the learning rate.

Due to the restricted nature of search compared with the RNN model, we were able to exhaustively search our hyperparameters. However, to combat the possible validation overfitting, we randomly omitted half of our validation every epoch per trial.

Note: Our 1st iteration (with no early stopping) of search had a grid of

$$H = (\gamma, \lambda, \alpha, \text{epochs}{=}100)$$

$$\gamma = \{\text{bert-}\{\text{base}, \text{large}\}\text{-}\{\text{uncased}, \text{cased}\}\},$$

$$\lambda \in \{0, 0.005, 0.01\},$$

$$\alpha \in \{5 \times 10^{-5}, 1 \times 10^{-5}, 5 \times 10^{-6}\}.$$

2.4.3 Compute

The entire grid search took about 12 hours, but due to power constraints such as the laptop running out of power (despite being charged), the total search time was actually just under 26 hours.

The training was run on a 14" unbinned M4 Max Macbook Pro with 128gb of RAM, connected to a 100W charger (which dropped wattage due to overheating issues). The fans on the laptop were manually set to 93% (7500 rpm) in order to prevent overheating on the laptop (due to the conservative default fan curve).

2.4.4 Difficulties and Insights

Training a transformer demands serious compute. The pre-trained models that we fine-tuned were very large, and the amount of time it took to run either grid search was considerable (in fact, the first grid search with no early stopping took over 50 hours).

In fact, the sheer size of the transformers suggests that it may be possible that even models as small as 100 million parameters may be too large, especially for a training dataset of 8000 samples. This may suggest that smaller models between 0.1 million to 10 million models may be advantageous.

Due to how much power the laptop draws while training the transformer, it had to be constantly connected to a charger, where the battery may drain at a rate of 1% per hour if the charger outputted 100w, and faster if the charger started to throttle. This rendered both the laptop useless, minorly inconveniencing one of the students.

2.4.5 Performance

MAE L_δ	**Model Name** γ	**Weight Decay** λ	**Learning Rate** α	**Training Duration**
4.440	distilbert-base-cased	0.005	5E-05	06:16
4.487	bert-base-cased	0.005	5E-05	17:43
4.932	distilbert-base-uncased	0.01	1E-04	03:16
5.040	distilbert-base-uncased	0	5E-05	04:26
5.135	bert-base-uncased	0.01	1E-04	09:55

The top 5 models from grid-searching. The full table is available in the Appendix.

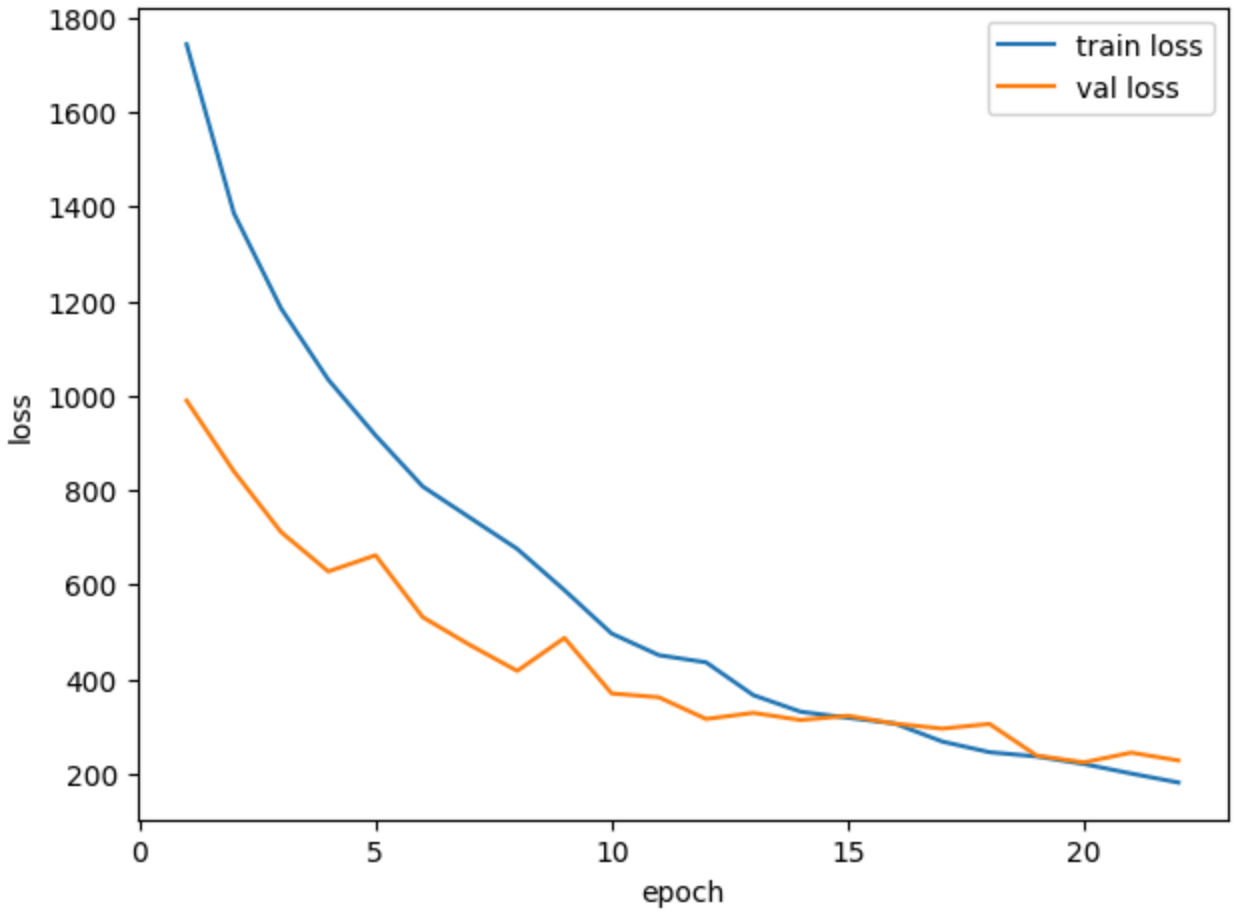
We achieve our best MAE of 4.44 with $\gamma = \text{distilbert-base-cased}, \lambda = 0.005, \alpha = 1 \times 10^{-5}$. Notably, our top performances are all pretty consistent, indicating that no specific model choice affects the performance much, as long as if hyperparameters are suitable for learning (except for Roberta, which has very poor performance).

Let us denote the top model by m_t^*. We have the following metrics:

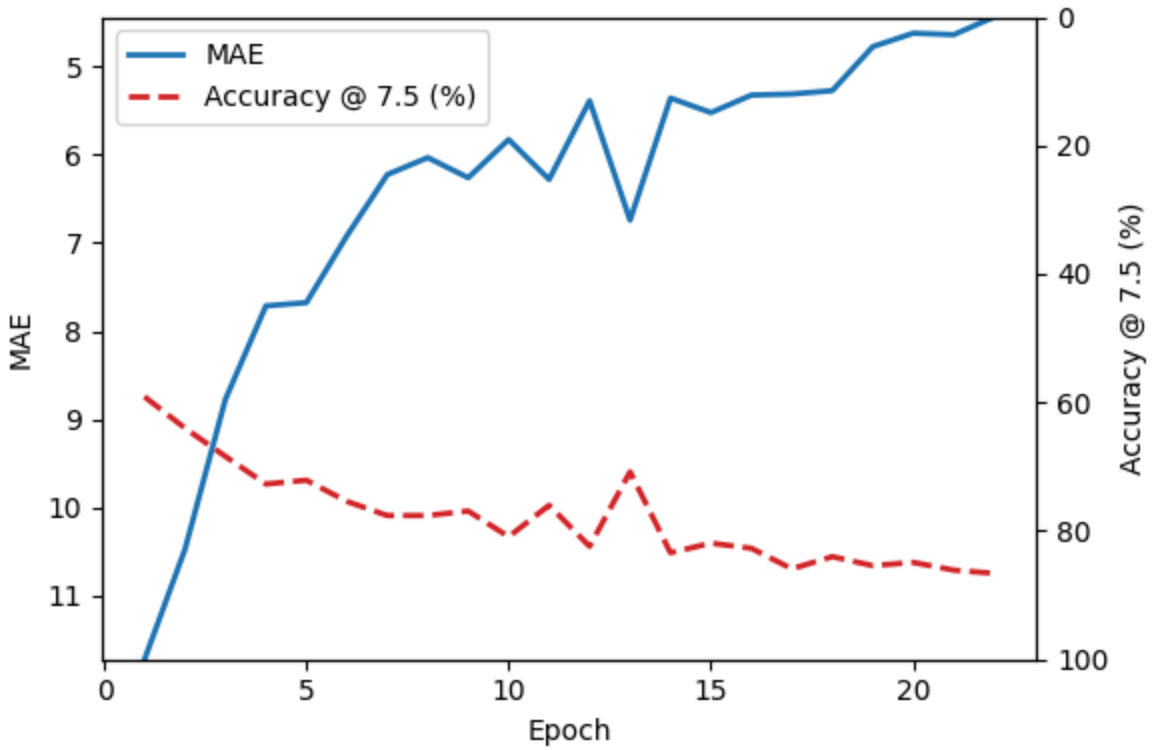
$$\tilde{L}_\delta(m_t^*) = 4.44$$

$$\tilde{L}_\beta(m_t^*) = 0.866$$

Additionally, the training of the top model indicates that the model should have quite generalizable performance (given little out-of-vocabulary errors).



Both the training and validation losses decrease throughout training, though the train loss decreases faster than the validation loss. However, since the validation loss does not generally increase, we can conclude that there is no significant overfitting of the model. Some interesting behaviors of training with respect to epochs and test performance is shown in a further section.



The accuracy of the model increases steadily throughout training, from about 60% initially to a final accuracy of 86.45%. Additionally, the MAE improves quickly at first, with improvements past 10 epochs, or about 6 MAE slowing down slightly.

2.4.6 Discussion

We can see that the top 5 models all perform better than SOTA, and have even lower validation MAE scores than the RNN model. Furthermore, all of the models except `bert-base-cased` stopped in under 10 minutes to train, indicating that the threshold to overfit is within 50 epochs for these models, though could be subject to change given that a linear decay of the learning rate was used.

Query s	**Carbs** c_s	**Prediction** $\hat{c}$	**Error** L_δ	**Relative Error** $L_\mathcal{R}$
One complete recipe of praline candies.	540.4813	116.4224	424.0589	0.7846
One whole strawberry pie.	707.52	397.5310	309.989	0.43813
A medium pizza with extra cheese and a thin crust.	19.602	205.7323	186.130	9.4955
A French bread pizza on thick crust, topped with cheese and extra vegetables.	50.3424	221.5759	171.233	3.4014
1 cup of simple syrup.	145.856	270.2931	124.437	0.8532

Query s	**Carbs** c_s	**Prediction** $\hat{c}$	**Error** L_δ	**Relative Error** $L_\mathcal{R}$
One package of TWIX Peanut Butter Cookie Bars.	144.5538	24.5099	120.044	0.8304
A cup of gumdrops.	179.998	60.2406	119.7574	0.6653
A 2-layer white cake made from a recipe without frosting.	507.364	389.3012	118.0628	0.2327
One extra-large gluten-free thin crust cheese pizza.	339.0534	231.9195	107.1339	0.316
One cup of oil-popped popcorn without butter.	102.4299	2.484	99.9459	0.9757

10 highest error queries

With the exception of the third query, all of the highest L_δ queries' ground truths have a generally high carb count, which is to be expected. However, we can see that the relative error is still relative large for all of these.

These queries all exhibit some sort of trickiness. For example, the top query shows has a quantifier of "One complete recipe," but that is almost universally vague. Further, praline candies are also pretty niche, so it's likely that this query induced two errors, which, when combined with the magnitude of what the carbs should be, creates the largest error L_δ. We can see similar behaviors in other top queries like the 6th query, which uses the quantifier "One package," though that is quite vague without having outside information as to what exactly "One package" means in relation to "TWIX Peanut Butter Cookie Bars."

Finally, there are simply vague descriptions that are in a large carb magnitude area, such as the 8th query, which simply does not have an exact quantifier besides the fact that the cake is "2-layer." Even for experts, it's not possible to say exactly large the cake would be, much less the carb count.

Query s	**Carbs** c_s	**Prediction** $\hat{c}$	**Error** L_δ	**Relative Error** $L_\mathcal{R}$
A medium pizza with extra cheese and a thin crust.	19.602	205.7323	186.1303	9.4955
An iced decaffeinated cafe mocha.	2.7869	21.4246	18.63772	6.6876
One clam cake.	8.148	61.1814	53.03342	6.5088
A cup of coffee substitute.	4.68	29.2217	24.54169	5.244

Query s	Carbs c_s	Prediction $\hat{c}$	Error L_δ	Relative Error $L_\mathcal{R}$
1 fl oz of 100% lemon juice served with ice.	1.2926	6.8438	5.5512	4.2946
1 fl oz of 100% lime juice served without ice.	2.6102	13.6925	11.0823	4.2458
French fries with chili.	2.5264	12.1997	9.6733	3.8289
One hard gluten-free flavored pretzel.	1.001	4.6139	3.6039	3.5683
A French bread pizza on thick crust, topped with cheese and extra vegetables.	50.3424	221.5759	171.2335	3.4014
One cup of Kimchi soup.	1.96	8.1955	6.2355	3.1814

10 highest relative error queries with $c_s > 1$.

For the top 10 relative error queries $L_\mathcal{R}$ excluding $c_s \leq 1$, we have errors ranging from 3 to 10. We can see that this time, the top queries have generally smaller ground truths, with only 2 outliers where $c_s > 10$. Note that the top query is one of these outliers and in fact is also the 3rd top query for L_δ. Consider this top query s^*.

Note that it is not possible for an entire pizza to contain only about 20 grams of carbs. Therefore, the query is implicitly alluding to a *single slice of pizza,* rather than the entire pizza itself. Therefore, if we consider that the pizza was cut into a standard configuration such as 6 or 8 slices, we would get much smaller losses, $L_\delta = 14.6867$ and $L_\mathcal{R} = 0.7492$, and $L_\delta = 6.1145$ and $L_\mathcal{R} = 0.3119$, placing it within most of the data. Because without the ground truth such a difference is indiscernible, such a query is unreasonable to judge the model with.

Consider the second top query. Whereas $c_s = 2.7869$, top websites including a regular starbucks order of nearly the same name has 29 grams for the smallest possible order. In fact, no source online states that an iced decaffeinated cafe mocha is less than 10 grams of carbs, indicating that the model indeed makes quite an accurate estimation. However, it is possible that the query is alluding to a very tiny cup of coffee (though highly unlikely and unreasonable without further clarification), in which case the ground truth would make more sense. However, because this is also impossible to realize without knowing the ground truth beforehand, this query is also unreasonable to judge the model with.

Consider the third top query. In this case, the ground truth is quite reasonable (if not slightly low), and the model fails to even be in the area of magnitude of carbs. While the training data has a very similar scenario, it's possible that due to the fact that the training query has the word "single" instead of "one," it's possible the model misinterpreted the "single" to indicate one of

many, whereas "one" may more likely indicate the entirety of something (of which it would give high attention to the word "cake", which generally has a carb count of about 60 grams).

3. Interesting notes

As alluded to in the exploratory analysis it appears that the test data may not be representative of the data that was used to train the model, or is much more challenging. In fact, our top RNN and Transformer models that achieved SOTA-beating validation performance do not generalize to the test data, as manually confirmed by Andong Hua. Despite much analysis into the architecture, we continually are unable to attribute this difference to anything from the models, but we hypothesize that it may be attributed to the difference in the test data from the train and validation data, both in the distribution of the carbs to the words in the queries.

Additionally, much of our analysis in the discussion sections seems to indicate that there are a lot of queries that are nearly impossible to decipher, and much of the data includes vague descriptors that negatively impact the models' ability to accurately predict the carb counts.

4. Conclusion

This study demonstrates that careful model selection and hyper-parameter optimization enable lightweight architectures to rival, and in validation terms outperform, state-of-the-art LLM solutions on the Nutribench carbohydrate-estimation task. The less than 1 million-parameter RNN achieves a 64% reduction in MAE relative to linear baselines, while a 66 million-parameter DistilBERT pushes the error below 5 grams within six minutes of fine-tuning. Qualitative error analysis highlights two persistent challenges: (1) ambiguous or underspecified quantity descriptors (“one slice”, “package”, etc.), and (2) distribution shift between training and test queries, the latter containing longer, bimodal descriptions. Addressing these issues—e.g., by incorporating quantity-normalization heuristics or data augmentation targeted at long-tail phrasing—offers a clear path toward further gains. More broadly, our results caution against defaulting to ever-larger models: when task scope is narrow and evaluation is numeric, compact networks not only suffice but can be preferable in terms of reproducibility, determinism, and energy footprint. Future work will extend this inquiry to multi-nutrient prediction and assess robustness under deliberate adversarial phrasing.

5. Appendix

Full Explanatory data

	Train	Validation	Test
Mean description length $\mu_{\text{Description length}}$	11.61	8.82	18.01
Vocab Size $\|S_V\|$	4816	2370	3294

	Train	Validation	Test
Mean Carbs μ_c	19.63	20.15	$-$
Standard Deviation of Carbs σ_c	42.94	39.36	$-$

Full RNN Grid Search Results

MAE L_δ	**Embedding Dimension** d_e	**LSTM Units** d_l	**RNN units** d_r	**Dropout** λ	**Learning Rate** α	**Duration**
9.274	256	32	22	0.238	2.284E-04	02:21
7.681	64	16	78	0.232	2.195E-03	02:19
9.160	256	64	22	0.455	8.240E-04	02:44
11.857	64	64	41	0.249	4.684E-03	02:22
8.533	256	16	56	0.208	5.386E-04	02:33
8.295	256	64	24	0.337	5.579E-04	02:41
9.112	256	64	38	0.470	8.108E-04	02:54
7.650	256	64	94	0.472	5.790E-04	03:49
—	256	32	99	0.400	5.498E-03	03:33
8.894	64	32	23	0.244	6.183E-04	04:12
8.730	256	16	45	0.471	5.730E-04	02:34
9.097	128	64	111	0.400	1.157E-04	03:54
8.224	64	16	92	0.326	2.571E-03	02:26
9.288	128	16	79	0.398	2.355E-03	02:39
20.972	64	16	108	0.287	8.669E-03	02:37
7.562	64	32	128	0.399	1.796E-03	03:03
7.926	128	32	125	0.414	1.363E-03	03:25
8.569	64	32	127	0.435	3.012E-04	03:07
9.736	256	32	103	0.496	1.323E-03	03:51
8.409	64	64	90	0.367	3.385E-04	03:12
15.336	128	32	65	0.373	4.298E-03	02:46
9.119	256	64	115	0.435	1.172E-04	04:29
7.799	64	16	74	0.296	2.012E-03	02:18
7.996	64	16	89	0.495	1.425E-03	02:26
9.064	64	32	100	0.307	3.711E-03	02:48

MAE L_δ	Embedding Dimension d_e	LSTM Units d_l	RNN units d_r	Dropout λ	Learning Rate α	Duration
19.503	64	16	62	0.363	7.028E-03	02:11
8.661	64	64	77	0.427	1.105E-03	03:01
8.702	64	32	82	0.200	1.966E-03	02:36
9.074	64	16	120	0.269	2.868E-03	02:48
20.672	128	64	96	0.458	6.109E-03	03:37

Note: $H = (d_e = 256, d_l = 32, d_r = 99, \lambda = 0.4, \alpha = 5.498 \times 10^{-3})$ has no L_δ due to being a failed trial and reporting an L_δ of `nan`*.*

Full Encoder-Transformer Grid Search Results

MAE L_δ	Model Name γ	Weight Decay λ	Learning Rate α	Training Duration
4.440	distilbert-base-cased	0.005	5E-05	06:16
4.487	bert-base-cased	0.005	5E-05	17:43
4.932	distilbert-base-uncased	0.01	1E-04	03:16
5.040	distilbert-base-uncased	0	5E-05	04:26
5.135	bert-base-uncased	0.01	1E-04	09:55
5.212	bert-base-cased	0	5E-05	12:49
5.241	distilbert-base-cased	0	5E-05	03:29
5.306	distilbert-base-uncased	0.005	1E-04	04:04
5.349	bert-base-cased	0.005	1E-04	13:50
5.396	bert-base-uncased	0.005	5E-05	12:11
5.408	bert-base-uncased	0.01	5E-05	15:31
5.578	bert-base-cased	0.01	5E-05	10:52
5.741	roberta-base	0.01	5E-05	12:54
5.893	distilbert-base-uncased	0.01	5E-05	03:30
6.090	bert-base-uncased	0	5E-05	15:26

MAE L_δ	Model Name γ	Weight Decay λ	Learning Rate α	Training Duration
6.166	distilbert-base-uncased	0.005	5E-05	03:14
6.253	distilbert-base-cased	0.01	5E-05	02:55
6.331	bert-base-cased	0	1E-04	-
6.502	distilbert-base-uncased	0	1E-04	03:02
6.561	distilbert-base-cased	0.005	1E-05	05:01
6.666	distilbert-base-uncased	0.01	5E-06	10:04
6.681	bert-base-cased	0.005	1E-05	34:16
6.698	distilbert-base-uncased	0.005	5E-06	10:26
7.107	distilbert-base-uncased	0.01	1E-05	05:31
7.154	distilbert-base-cased	0.01	1E-05	06:16
7.207	distilbert-base-uncased	0	1E-05	05:29
7.242	distilbert-base-cased	0	1E-05	08:06
7.378	distilbert-base-cased	0	1E-04	02:37
7.487	roberta-base	0	5E-05	09:10
7.510	distilbert-base-uncased	0.005	1E-05	05:31
7.604	roberta-base	0.005	5E-05	10:30
7.682	distilbert-base-cased	0.005	1E-04	02:53
7.765	distilbert-base-cased	0.01	1E-04	02:55
8.079	bert-base-uncased	0	1E-04	06:49
8.145	bert-base-uncased	0	1E-05	14:01
8.210	bert-base-uncased	0.005	1E-05	21:57
8.280	bert-base-cased	0	1E-05	24:38
8.360	distilbert-base-uncased	0	5E-06	10:40
8.366	bert-base-cased	0.01	1E-04	06:26
8.702	bert-base-cased	0.01	1E-05	11:55

MAE L_δ	Model Name γ	Weight Decay λ	Learning Rate α	Training Duration
8.817	roberta-base	0.005	1E-05	22:20
9.008	bert-base-uncased	0.005	1E-04	04:46
9.104	distilbert-base-cased	0	5E-06	06:32
9.162	roberta-base	0.01	1E-05	19:14
9.226	bert-base-uncased	0.01	1E-05	18:46
9.554	roberta-base	0	1E-05	15:19
9.811	bert-base-cased	0	5E-06	21:37
10.155	roberta-base	0.01	5E-06	17:11
10.244	bert-base-uncased	0.01	5E-06	23:30
10.532	bert-base-uncased	0	5E-06	21:05
10.753	roberta-base	0.005	5E-06	23:51
10.903	bert-base-cased	0.005	5E-06	09:55
10.941	distilbert-base-cased	0.005	5E-06	04:12
11.041	bert-base-uncased	0.005	5E-06	23:39
11.183	distilbert-base-cased	0.01	5E-06	04:03
11.266	roberta-base	0	5E-06	18:33
12.147	bert-base-cased	0.01	5E-06	15:17
19.214	bert-base-uncased	0.005	5E-04	02:51
19.523	bert-base-cased	0.005	5E-04	02:59
19.714	bert-base-cased	0	5E-04	03:09
20.099	roberta-base	0.005	5E-04	03:22
20.140	roberta-base	0	1E-04	01:57
20.491	roberta-base	0.01	1E-04	01:26
20.502	roberta-base	0.005	1E-04	03:50
20.518	distilbert-base-uncased	0.005	5E-04	00:47
20.530	bert-base-uncased	0	5E-04	02:51
20.689	distilbert-base-uncased	0.01	5E-04	00:47
21.112	roberta-base	0	5E-04	03:24
21.207	bert-base-cased	0.01	5E-04	02:59

MAE L_δ	**Model Name** γ	**Weight Decay** λ	**Learning Rate** α	**Training Duration**
21.234	roberta-base	0.01	5E-04	03:21
21.288	distilbert-base-cased	0.01	5E-04	00:50
21.323	bert-base-uncased	0.01	5E-04	02:52
21.439	distilbert-base-cased	0.005	5E-04	00:50
21.646	distilbert-base-uncased	0	5E-04	00:47
21.880	distilbert-base-cased	0	5E-04	00:49

Note: $H = (\gamma = bert\text{-}base\text{-}cased, \lambda = 0, \alpha = 1 \times 10^{-4}, epochs{=}100)$ *has no given training time due to loss of power during training*