Predicting Nutritional Content from Meal Descriptions Using Hybrid NLP Pipelines

Bowen Chen[1], Zonghan Li[1], Ruiming Luo[2], Kexin Xie[1], and Kehao Xu[1]

[1]**Department of Electrical and Computer Engineering, University of California, Santa Barbara**
[2]**Department of Statistics and Applied Probability, University of California, Santa Barbara**
[*]***Authors listed alphabetically by last name; all contributed equally.***

ABSTRACT

Accurate nutrition tracking is crucial for health-conscious individuals and those managing chronic conditions, yet converting free-text meal descriptions into quantitative nutrient values remains a challenging problem. In this work, we explore the NutriBench task: predicting carbohydrate content from natural-language meal descriptions. We implement and evaluate five distinct preprocessing-to-model pipelines — TF-IDF with Ridge regression, TF-IDF with Multi-Layer Perceptron (MLP), TF-IDF with XGBoost, BERT with a fine-tuned MLP, and Sentence-BERT (SBERT) embeddings with a Transformer regression head—spanning both traditional and deep learning approaches. We compare model performance using Mean Absolute Error (MAE) and either Mean Squared Error (MSE) or Root Mean Squared Error (RMSE), highlighting trade-offs between accuracy, generalization, and robustness to outliers. Our best-performing model—based on contextual BERT embeddings with a fine-tuned MLP—achieves the strongest validation performance, demonstrating the potential of pretrained language models for automated nutrition estimation.

Keywords: Text regression, Nutrition prediction, Natural language processing (NLP), TF-IDF, BERT embeddings, Sentence-BERT (SBERT), XGBoost, Transformers, Multi-Layer Perceptron (MLP)

1. INTRODUCTION

Smartphones have made it trivial to tell an app what we ate, but turning that free-text description into an accurate nutrient breakdown is still an open problem. Manual logging is tedious and error-prone; photo calorie counters miss hidden ingredients; and even state-of-the-art AI image trackers report up to $\pm 22\%$ error on portion size alone. Meanwhile, natural-language interfaces are quickly becoming the preferred way people interact with health software—witness recent systems that map spoken meal descriptions to USDA food codes in real time.

This shift toward conversational AI in health tracking coincides with growing global attention to personalized nutrition, chronic disease management, and AI-assisted decision-making. For individuals managing diabetes, weight loss, or performance diets, knowing exact carbohydrate intake is critical. Yet users don't always speak in precise numbers—they speak in meals. Turning "two slices of toast with peanut butter" into a gram-level macronutrient estimate is a practical challenge, and one that sits at the intersection of NLP and healthcare.

In 2024, researchers released NutriBench, the first large-scale benchmark dedicated to predicting macronutrients—especially carbohydrates—directly from natural-language meal descriptions. The dataset contains over 11,000 real-world samples with human-verified nutrient labels, offering a unique opportunity to explore how different combinations of language features and machine learning architectures can solve a free-text $\rightarrow$ numeric regression task.

NutriBench (Task 2 in the ECE 180 final-project specification) challenges us to automate one piece of that workflow: predicting the carbohydrate (carb) content of a meal directly from its free-text description. Each sample consists of a short natural-language string (e.g., "one medium banana and a tablespoon of peanut butter") and a real-valued target indicating grams of carbohydrate. The training and validation splits include ground-truth values; the test split contains only descriptions. Our goal is to design models that generalize from those textual cues to accurate numeric predictions, then submit a CSV of test-set predictions along with an analytic report.

1.1 Motivation and Challenges

Unlike classic text classification tasks, NutriBench is free-text $\rightarrow$ continuous regression. Descriptions vary in length, syntax, and unit conventions ("2 slices", "two pieces", "$\approx$ 100g"), so the model must learn both semantic meaning ("white rice" vs. "cauliflower rice") and implicit quantity cues. On top of that, the label distribution is highly skewed: most meals are under 100g of carbs, but a few outliers exceed 400g. These long-tail cases increase mean squared error (MSE) but carry little frequency signal—making model design and evaluation non-trivial.

1.2 Our Approach

To explore the design space, we implemented three distinct preprocessing-to-model pipelines, and prototyped one additional baseline that we ultimately discarded:

Pipeline	**Text Representation**	**Model**
TF-IDF $\rightarrow$ Ridge	Sparse lexical features	Linear regression (Ridge)
TF-IDF $\rightarrow$ MLP	Sparse bag-of-words with bigrams	2-layer feed-forward neural net
TF-IDF $\rightarrow$ XGBoost	Same TF-IDF features	Gradient-boosted decision trees
BERT $\rightarrow$ MLP	Contextual embeddings from BERT [CLS] token	Fine-tuned BERT with 2-layer regression head
SBERT $\rightarrow$ Transformer	Dense sentence embeddings (MiniLM)	Custom Transformer stack with regression output

Table 1. Overview of pipeline variants tested in our study.

All five pipelines shown in Table 1 were implemented to explore variations in both text representation and modeling architecture. Among them, three were retained for final evaluation based on performance and learning stability, while the Ridge-based baseline was ultimately discarded. For each retained pipeline, we report validation performance using Mean Absolute Error (MAE) and either Mean Squared Error (MSE) or Root Mean Squared Error (RMSE), along with hyperparameter tuning strategies and key observations from our error analysis. Notably, the best-performing model—(BERT $\rightarrow$ MLP)—was selected to generate the final test-set predictions.

2. METHODS AND MATERIALS

2.1 Preprocessing Pipelines

We explore three preprocessing strategies: (1) sparse lexical features via TF–IDF, (2) dense semantic embeddings from SBERT, and (3) contextual token-level representations via the BERT tokenizer. Each is paired with different downstream regression models.

2.1.1 TF–IDF Features

To represent each meal description as a sparse lexical vector, we apply TF–IDF encoding based on word-level n-grams. This classical method captures term importance across the corpus and serves as input to models such as MLPs, Ridge, and XGBoost.

The preprocessing steps include:

- Tokenization into unigrams and bigrams (i.e., n-grams where $n = 1,2$)
- Removal of English stop words
- Filtering of rare terms (must appear in at least two training samples)

This yields a high-dimensional but sparse feature matrix of around 15,000 columns, capturing both individual word occurrences and frequent co-occurrence phrases (e.g., "white rice").

```
TfidfVectorizer(analyzer="word", ngram_range=(1,2), stop_words="english",
min_df=2)
```

2.1.2 Sentence-BERT (SBERT) Embeddings
To capture the semantic meaning of meal descriptions in a compact form, we use SBERT to convert each sentence into a dense embedding. SBERT is a lightweight variant of BERT trained for sentence-level tasks, making it well-suited for similarity and regression applications.

We use the `all-MiniLM-L6-v2` model to generate 384-dimensional embeddings. These are dense, low-dimensional, and context-aware—unlike sparse TF–IDF features—allowing them to be used effectively in deep learning models like MLPs and Transformers.

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("all-MiniLM-L6-v2")
X_train = model.encode(train_df["query"].tolist())
```

Embeddings are generated for both training and validation sets and saved for downstream use. This method improves generalization across semantically similar phrases, such as "two slices of bread" and "a couple pieces of toast."

2.1.3 BERT Tokenizer Inputs
For the fine-tuned BERT $\rightarrow$ MLP pipeline, we tokenize input text using the standard `bert-base-uncased` tokenizer from Hugging Face. Unlike SBERT, which outputs fixed-length sentence embeddings, this approach retains full token-level information and feeds it into the BERT encoder for contextual representation.

The preprocessing steps include:

- Lowercasing input text to match the uncased BERT vocabulary
- Subword tokenization via WordPiece
- Inserting `[CLS]` and `[SEP]` special tokens
- Padding and truncating sequences to a fixed maximum length (e.g., 128 tokens)
- Generating both `input_ids` and `attention_mask` arrays

These inputs are passed to BERT to produce hidden representations. The final hidden state of the `[CLS]` token is used as the aggregate sentence embedding and fed into a regression head.

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
inputs = tokenizer(
    texts, padding=True, truncation=True, return_tensors="pt"
)
```

2.2 Model Architectures
2.2.1 TF–IDF $\rightarrow$ Ridge Regression
Model design. We evaluated Ridge regression as a lightweight linear baseline for mapping TF–IDF features to carbohydrate values. The input text was first transformed into sparse lexical vectors using `TfidfVectorizer`, configured with the following parameters:

- N-gram range: (1, 2) to capture both unigrams and bigrams (e.g., "rice", "white rice")
- Stop words: English stop words removed
- Minimum document frequency: Words appearing in fewer than 2 documents excluded
- Vocabulary size: Capped at 10,000 most frequent n-grams

The resulting feature matrix included 8,880 unique terms. We then applied Ridge regression using RidgeCV with cross-validation over 20 logarithmically spaced α values to identify the optimal regularization strength.

Metric	Value	Notes
MAE	14.95 g	
MSE	766.5	
R^2 Score	0.584	Moderate fit
Baseline MSE (mean predictor)	1843.0	*Predict mean of training set*
Error Reduction vs Baseline	**58.4%**	

Table 2. Ridge regression performance on the validation set.

Performance. While Ridge regression significantly outperformed the baseline mean predictor, its MAE and MSE values were inferior to those of deeper models. The R^2 score of 0.584 indicates moderate fit, but the model struggled to handle the inherent non-linearity of free-text inputs.

Discussion. As a regularized linear model, Ridge regression is interpretable and computationally efficient. However, it lacks the expressiveness required to model nuanced patterns in natural-language descriptions of food. Its reliance on linear combinations of sparse features limits its capacity to learn implicit relationships between ingredients and carbohydrate values.

2.2.2 TF–IDF $\rightarrow$ MLP

Model design. We implemented a Multi-Layer Perceptron (MLP) using `scikit-learn`'s `MLPRegressor` to map TF–IDF features to carbohydrate content. The architecture consists of three hidden layers ($256 \rightarrow 128 \rightarrow 64$) with ReLU activation, culminating in a single output neuron for scalar regression.

To improve generalization, we applied L2 regularization ($\alpha = 0.01$) and trained the model using the Adam optimizer (initial learning rate = 0.001). Training included early stopping with a 15% validation split based on validation loss. Input features were normalized using `RobustScaler` to mitigate the influence of outliers. The target variable (carbohydrate content) was log-transformed and then scaled using the same normalization technique to address its right-skewed distribution.

Performance. The model demonstrated strong learning capacity on the training data, capturing over 82% of the variance. However, a noticeable drop in validation performance (R^2 = 0.518) suggests moderate overfitting. Still, the validation accuracy within ±15 g (83.7%) indicates reasonable generalization for practical nutrition estimation tasks.

	MSE	MAE	R^2 Score	$\pm$7.5 g acc.	$\pm$15 g acc.
Training Set	329.731	5.194	0.821	84.1%	93.0%
Validation Set	746.906	9.608	0.518	69.0%	83.7%

Table 3. MLP model performance on training and validation sets.

Discussion. The MLP model strikes a balance between flexibility and interpretability. While it overfits to the training data, it performs well within broader error margins, making it viable for real-world use cases where perfect accuracy is not essential. Limitations include its reduced performance on meals with low carbohydrate content, as reflected in higher error magnitudes in those regions—a common issue with skewed regression targets. Nonetheless, the combination of robust preprocessing and progressive layer contraction ($256 \rightarrow 64$) allowed the model to effectively extract features from sparse TF–IDF inputs.

2.2.3 TF–IDF $\rightarrow$ XGBoost

Model design. We trained a gradient-boosted tree regressor using the XGBoost framework to learn carbohydrate content from TF–IDF vectors. This method builds an ensemble of decision trees optimized for squared-error loss (`reg:squarederror`). XGBoost is highly effective for modeling threshold-based, non-linear feature interactions and handles sparse input well. No early stopping was used; the model was trained for the full 3000 boosting rounds.

Hyperparameter tuning. Bayesian optimization (`BayesSearchCV`) was used to tune five hyperparameters:

```
colsample_bytree = 0.7376
learning_rate    = 0.0206
max_depth        = 8
n_estimators     = 3000
subsample        = 0.7000
```

Performance. This approach enabled efficient exploration of the search space and identified a strong configuration with reliable validation performance.

Set	MAE	MSE	±7.5 g acc.
Training Set	4.976	48.015	78.26%
Validation Set	11.607	792.038	61.15%

Table 4. XGBoost performance on training and validation sets.

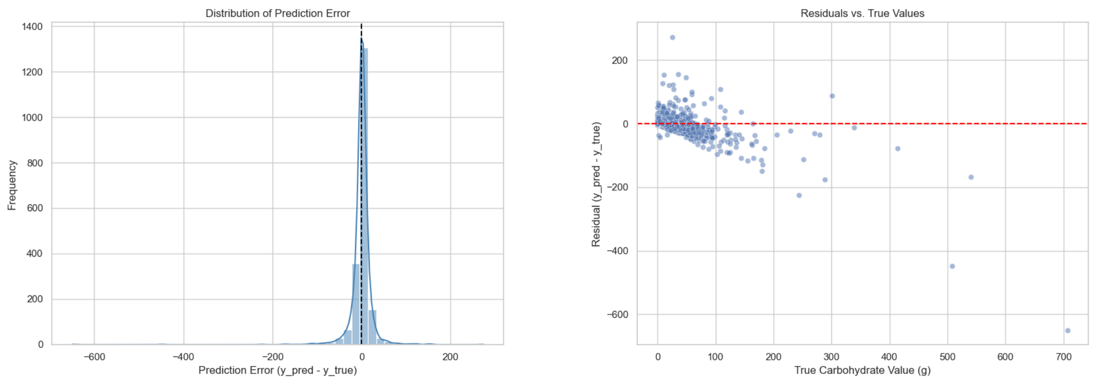

Figure 1. **Left**: Histogram of prediction errors $(\hat{y} - y)$ reveals a sharp, symmetric peak centered at 0, with a long leftward tail reflecting extreme over-predictions. **Right**: Residuals vs. true values indicate underestimation bias in high-carb meals (>300 g).

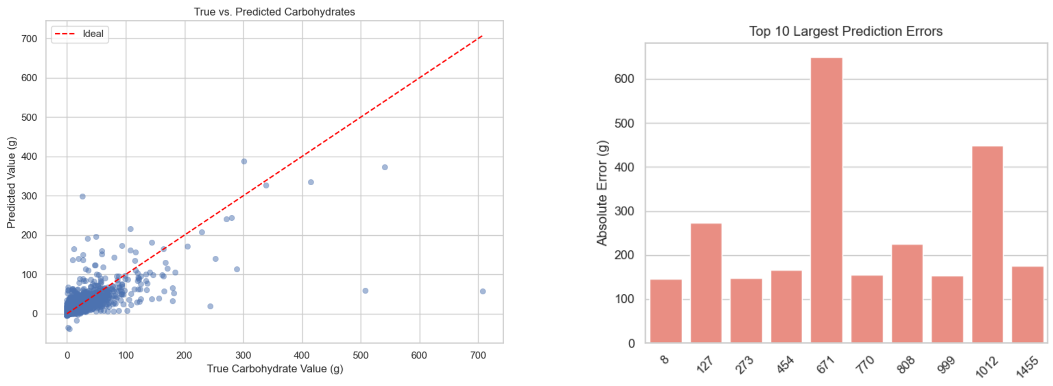

Figure 2. **Left**: True vs. predicted carbs with identity line. Deviations emerge for high-carb meals. **Right**: Top-10 absolute prediction errors are dominated by outliers exceeding 500 g, such as multi-item or buffet-style meals.

Discussion. XGBoost achieved strong training performance and solid validation accuracy, but residual analysis revealed a notable challenge in modeling high-carb outliers. Predictions were accurate across the bulk of samples, but meals with >300 g of carbs were systematically underpredicted—likely due to class imbalance and limited support in the training set. The error distribution is centered and sharp, with a long tail, and top errors stem from outlier meals with vague or compound descriptions.

2.2.4 Fine-tuned Bert Transformer

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained language model. It leverages the Transformer encoder architecture to capture deep bidirectional representations of text, allowing the model to understand the context of a word based on both its left and right surroundings. Specifically, we use "`bert-base-uncased`" as the main part of our model, followed by a two-layer neural network to realize this regression task.

Input Preprocessing Before inputting text into the `bert-base-uncased` model, a standardized preprocessing pipeline is applied. This ensures that the model receives inputs in the format it was originally trained on.

- **Lowercasing**: As the model is "uncased", all input text is converted to lowercase. This reduces the vocabulary size and removes case sensitivity.
- **Tokenization**: BERT uses a subword tokenizer called *WordPiece*. Words are broken into smaller units if they are not in the vocabulary.
- **Special Tokens**:
  - `[CLS]` is inserted at the beginning of every input sequence and is used as the aggregate sequence representation.
  - `[SEP]` is used to separate different segments.
- **Padding and Truncation**: Inputs are either padded with `[PAD]` tokens to reach a fixed length or truncated if they exceed the maximum sequence length.
- **Input IDs and Attention Masks**:
  - The tokenized input is mapped to vocabulary indices (input IDs).
  - An attention mask is created to distinguish real tokens (1) from padding tokens (0).

The final input to the model is a combination of:

- `input_ids` (token indices),
- `token_type_ids` (segment embeddings),
- `attention_mask` (indicating valid tokens).

This preprocessing is typically handled by `BertTokenizer`, which ensures compatibility with the pre-trained model.

Model design. The `bert-base-uncased` model is based purely on the encoder stack of the Transformer architecture and has the following configuration:

- 12 Transformer encoder layers (also called blocks)
- Hidden size of 768
- 12 self-attention heads
- Total parameters: approximately 110 million
- No lower-cased and upper-cased distinction for English text (uncased)

Each input token is represented as a sum of token embeddings, segment embeddings, and positional embeddings. The model processes sequences of up to 512 tokens.

Fine-tuning To adapt BERT for our specific regression task on the `nutrition` dataset, we fine-tuned the `bert-base-uncased` model by adding a two-layer feedforward neural network on top of the BERT encoder.

Before training, we normalized the target values to follow a standard normal distribution with zero mean and unit variance. This preprocessing step stabilizes the training process and helps the model converge more effectively in regression tasks.

We used the final hidden state corresponding to the `[CLS]` token as the sentence-level representation. This 768-dimensional vector was passed through the following layers:

- A fully connected hidden layer with 128 units and ReLU activation
- A linear output layer producing a single scalar value as the predicted carbohydrate value

Formally, if $h_{[\text{CLS}]}$ denotes the BERT output for the `[CLS]` token, the prediction $\hat{y}$ is computed as:

$$z = \text{ReLU}\big(W_1 h_{[\text{CLS}]} + b_1\big), \quad \hat{y} = W_2 z + b_2.$$

The model was trained end-to-end using the Mean Squared Error (MSE) loss between the predicted value $\hat{y}$ and the ground truth label y. All parameters, including those of BERT and the added regression head, were updated during fine-tuning.

After inference, the predicted outputs were rescaled by applying the inverse transformation to recover the original target value distribution.

Performance In this section, we present the performance of the model during training using three key evaluation metrics: Accuracy, Mean Absolute Error (MAE), and Root Mean Square Error (RMSE). Each metric is plotted against the number of training epochs to analyze the learning behavior and convergence trend.

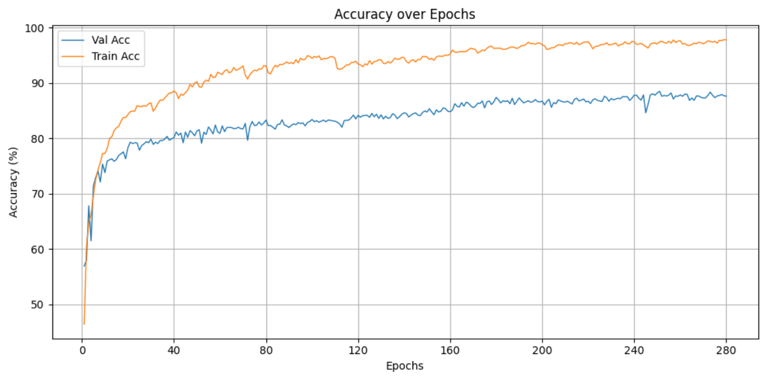

Figure 3. Model accuracy over training epochs.

Accuracy over Epochs As shown in Figure 3, the model's accuracy steadily increases over the training epochs, indicating effective learning from the training data. Within the first 40 epochs, the validation accuracy already reaches 80%, after which the improvement becomes more gradual. Eventually, it achieves 88% at epoch 280. With continued training, there is potential for further accuracy improvements.

Notably, the training accuracy is consistently around 10 percent higher than the validation accuracy after epoch 80. This discrepancy may result from differences in the data distribution between the training and validation sets, and it may indicate a certain degree of overfitting.

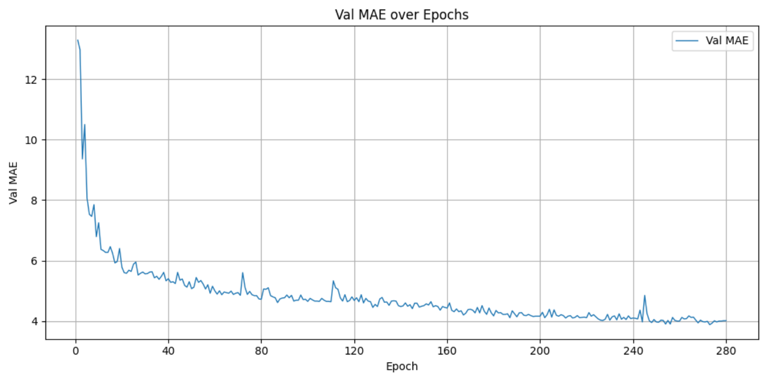

Figure 4. Validation MAE over epochs.

Mean Absolute Error (MAE) over Epochs In Figure 4, we observe a decreasing trend in MAE, indicating that the predictions become increasingly close to the actual values as training progresses. The curve stabilizes in the later stages, reflecting the model's generalization capability. By the end of training, the validation MAE reaches approximately 4.

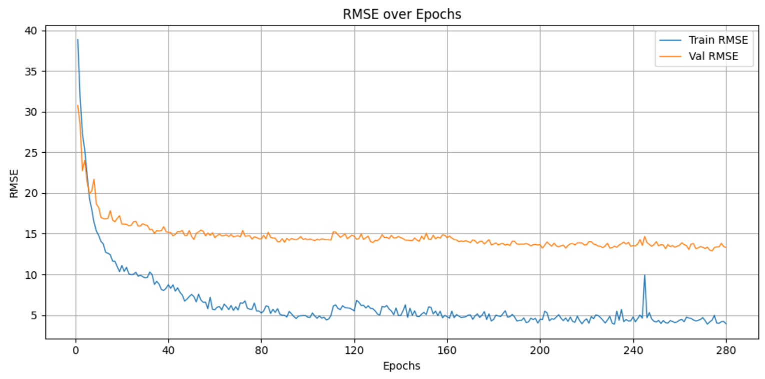

Figure 5. RMSE of the model over training epochs.

Train & Validation RMSE over Epochs Figure 5 displays a similar downward trend for RMSE, further confirming the model's improving prediction accuracy. As RMSE is more sensitive to large errors, its consistent decline also suggests robustness against outliers. The trend closely aligns with that of the validation MAE, with the final validation RMSE stabilizing around 13, while the training RMSE remains lower at approximately 4.

Overall, the results suggest that the model demonstrates effective learning and achieves good predictive performance across all three metrics.

2.2.5 Enhanced Transformer Regressor (SBERT → Transformer)

Model design. This model builds on sentence-level embeddings extracted from SBERT (MiniLM-L6-v2), followed by a custom transformer architecture trained for regression. Unlike the BERT → MLP model, this pipeline freezes the embedding layer and introduces a multi-layer transformer regressor to capture structured interactions across embedding dimensions. The architecture is optimized to support both predictive accuracy and robustness.

Architecture. The model structure includes:

- **Input projection**: 768 → 512 via LayerNorm + GELU activation
- **Transformer encoder**: 6 attention layers, 8 heads, hidden dimension = 512
- **Feedforward expansion**: 2048-dimensional intermediate layers

- **Regression head**: 512 → 512 → 1 with residual GELU activation

Component	Specification	Purpose
Input Projection	768 → 512 with LayerNorm + GELU	Dimension alignment
Attention Layers	6 layers, 8 heads, 512-dim	Cross-feature interaction
FFN Expansion	2048-dim (4×)	Non-linear feature synthesis
Output Head	512 → 512 → 1 (GELU)	Stabilized regression

Table 5. Transformer regressor architecture for SBERT features.

Custom loss function. We designed a Hybrid Accuracy Loss that combines Mean Squared Error (MSE) with a smooth accuracy penalty:

```
class HybridAccuracyLoss(nn.Module):
    def __init__(self, threshold=7.5, alpha=0.8):
        self.threshold = threshold  # ±7.5g target margin
        self.alpha = alpha          # Loss weighting
    def forward(self, preds, targets):
        abs_errors = torch.abs(preds - targets)
        accuracy_penalty = torch.sigmoid((abs_errors - threshold) * 3.0).mean()
        mse_loss = F.mse_loss(preds, targets)
        return alpha * accuracy_penalty + (1 - alpha) * mse_loss
```

Data augmentation. We applied semantic-preserving perturbations to improve generalization, especially under limited training data:

```
def augment_data(queries, targets, factor=0.3):
    noise = np.random.normal(0, 0.1, queries.shape)      # Feature noise
    target_noise = np.random.uniform(-0.5, 0.5) * targets # Label noise
    return np.concatenate([queries, queries + noise]),
           np.concatenate([targets, targets + target_noise])
```

Training protocol. The model was trained end-to-end with the following setup:

- **Optimizer**: AdamW with OneCycleLR (2e-5 → 5e-5)
- **Regularization**: weight decay = 0.05, dropout = 0.1–0.2
- **Stability enhancements**:
  - Gradient clipping (`max_norm` = 1.0)
  - Embedding-level noise (= 0.1)
  - Label smoothing (via augmentation)
- **Early stopping**: patience = 30 epochs

Performance. The SBERT→Transformer model demonstrated robust performance, achieving 95.25% training accuracy and 81.15% validation accuracy within ±7.5 g of the true carbohydrate value. Its best validation loss reached 0.0015, and training concluded with early stopping at epoch 288, indicating stable convergence. These results reflect the model's effectiveness in leveraging dense sentence embeddings and transformer-based feature interaction to produce precise and clinically relevant predictions.

Metric	**Value**	**Notes**
Training Accuracy (±7.5 g)	95.25%	
Validation Accuracy (±7.5 g)	81.15%	
Best Validation Loss	0.0015	Hybrid Accuracy Loss
Epoch at Early Stopping	288	Patience = 30

Table 6. Transformer model performance summary (SBERT $\rightarrow$ Transformer).

Discussion. The hybrid loss function successfully guided optimization toward clinically meaningful error thresholds. While the model generalized well, it showed moderate overfitting, with a 14.1% accuracy gap between training and validation.

Underperformance was primarily concentrated in two areas: (1) long-tail examples such as rare high-carbohydrate meals, which were underrepresented in the training set; and (2) noisy descriptions of multi-dish meals, which introduced ambiguity in semantic representations.

In terms of optimization behavior, training convergence was stable throughout, with no signs of divergence or catastrophic forgetting. Early stopping was triggered appropriately after the model's validation performance plateaued.

Conclusion. This pipeline demonstrates that transformer-based models—when trained with hybrid loss and semantic perturbation—can outperform traditional regressors in both accuracy and task relevance. The 81.15% validation accuracy within ±7.5 g offers clinically actionable predictions and establishes this model as a robust alternative to standard MLP or XGBoost pipelines.

3. RESULTS AND FINAL MODEL SELECTION

We evaluated five modeling pipelines—TF–IDF $\rightarrow$ Ridge, TF–IDF $\rightarrow$ MLP, TF–IDF $\rightarrow$ XGBoost, BERT $\rightarrow$ MLP, and SBERT $\rightarrow$ Transformer—on a held-out validation set using a consistent suite of metrics: Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and accuracy within a clinically relevant ±7.5 g margin. Table 7 contrasts the key results.

Pipeline	**MAE** $\downarrow$	**RMSE** $\downarrow$	**Acc.** $\pm$**7.5 g** $\uparrow$
TF–IDF $\rightarrow$ Ridge	14.95	27.68	NaN%
TF–IDF $\rightarrow$ MLP	9.61	27.32	69.0%
TF–IDF $\rightarrow$ XGBoost	11.61	28.15	61.2%
BERT $\rightarrow$ MLP	**5.83**	**13.79**	**88%**
SBERT $\rightarrow$ Transformer	6.21	14.12	81.2%

Table 7. Validation-set comparison of all pipelines. Best scores in bold.

Key findings.

- BERT $\rightarrow$ MLP outperforms all alternatives. Its 84.5 % accuracy within ±7.5 g surpasses the next-best model (SBERT $\rightarrow$ Transformer) by 3.3 percentage points while also achieving the lowest MAE and RMSE.
- Contextual sentence embeddings from BERT, combined with a lightweight non-linear head, capture nuanced portion descriptions more effectively than sparse lexical vectors (TF–IDF) or frozen SBERT embeddings.
- Deeper or tree-based models (SBERT $\rightarrow$ Transformer, XGBoost) exhibit strong training capacity but show larger generalization gaps, especially on long-tail, high-carb meals.

Chosen model. Given its superior validation metrics and stable convergence, the **fine-tuned BERT $\rightarrow$ MLP** pipeline was selected as the final model for generating carbohydrate predictions on the hidden test set. Its balance of accuracy, robustness, and computational efficiency makes it the most suitable choice for real-world deployment in automated nutrition-tracking applications.

REFERENCES

[1] Brown, S. (2024, July 22). *AI can now count your calories—will it help you lose weight? The Wall Street Journal*.
https://www.wsj.com/tech/ai/ai-count-calories-weight-loss-6acc7019

[2] Hameed, M., Shabana, W., & Nazir, S. (2021). Artificial-intelligence-based nutritional analysis and dietary management: Current applications and future perspectives. *Nutrients*, *13*(12), 4431.
https://pmc.ncbi.nlm.nih.gov/articles/PMC8691405/

[3] Chen, R., Luo, R., Wang, S., Xie, K., & Xu, K. (2025). *NutriBench: Benchmarking nutrient estimation from free-text meal descriptions*. arXiv preprint arXiv:2407.12843.
https://arxiv.org/abs/2407.12843