

Abstract

In 2023, 24.4% of clothes bought online in the US were returned (2023) [1], largely due to poor fit. This results in substantial economic and environmental costs. Extracting actionable fit data from unstructured customer reviews is paramount to mitigate this issue. Traditional sentiment analysis lacks the necessary aspect specific granularity. This project aims to develop, optimize, and evaluate a transformer-based Aspect Based Sentiment Analysis (ABSA) pipeline to accurately extract specific clothing aspects and classify their fit sentiment on an ordinal scale.

A two stage architecture was developed to achieve this, using the bidirectional bert-base-uncased model. Additionally, a local 20-billion parameter Large Language Model (LLM) was utilized to label data using Distillation. The models were then optimized, employing a sequential greedy search ablation study.

The optimized BERT pipeline fulfilled the project goals, achieving a system macro F1-score of 0.704, outperforming the baseline rule-based pipeline by 183.9%. The ordinal sentiment quantifier proved to be reliable, producing no extreme miscalculations between opposite fit sentiments (categorizing "Small" as "Large" or vice versa). This study concludes that Transformer based ABSA is an effective way to analyze the complex semantics of customer reviews, and that LLM Knowledge Distillation is an effective way to resolve data scarcity in specialized NLP domains. The developed ABSA pipeline provides a scalable, sustainable solution, providing consumers and businesses with product fit insights, reducing "bracket buying".

Keywords: *Aspect-Based Sentiment Analysis, Transformer Neural Networks, Ordinal Classification, Knowledge Distillation, Natural Language Processing, E-commerce Sustainability*

Table of contents

Abstract	3
1 Introduction and background	5
2 Literature Review	5
3 Methodology and Theory	7
3.1 Aspect Extractor Forward Pass	7
3.2 Aspect Extractor Backward Pass (Training)	11
3.3 Quantifier Forward Pass	14
3.4 Quantifier Backward Pass (Training)	15
3.5 Full ABSA Pipeline Summary	16
3.6 SpaCy and VADER NLP Baseline	16
4 Experimental Work and Investigation	18
4.1 Dataset Engineering	18
4.2 Evaluation Metrics	20
4.3 Experimental Setup and Ablation Study	23
5 Presentation of Results	24
5.1 Aspect Extractor Evaluation	25
5.2 Sentiment Quantifier Evaluation	28
5.3 Full ABSA Pipeline Evaluation	29
6 Discussion and Application	30
6.1 Economic and Environmental Sustainability	31
6.2 Discussion of Findings	32
6.3 System Limitations	32
6.4 Future Research	33
6.5 Conclusion	33
7 Project Management, Sustainability, and Risk	33
References	34

1 Introduction and background

The e-commerce industry generates an immense volume of unstructured customer review data. A challenge faced by many in the industry is the high rate of product returns (24.4% in the US 2023) [1] driven by incorrect clothing fitting. This fit related uncertainty drives "bracket buying" [2], causing financial and environmental sustainability issues. Customer feedback is a goldmine for information. However, analyzing this to develop actionable insights comes with complications. Traditional, rule-based NLP relies on predefined words and static grammatical rules, it frequently fails on complex syntax, sarcasm and ambiguity. Machine Learning (ML) approaches, transformers specifically, overcome this by dynamically weighing contextual relationships between tokens. This makes ML approaches far superior for extracting more nuanced insights.

This project aims to develop, optimize and evaluate transformer based ABSA able to accurately locate specific aspects related to clothing fit from unstructured reviews and classify their fit sentiment on an ordinal scale [3] (0: Small/Tight, 1: Perfect/True-To-Size , 2: Large/Loose, 3: Irrelevant).

This project uses a two-stage ML architecture. Taking advantage of the bidirectional bert-base-uncased model [4]. A typical bottleneck of domain specific labeled data scarcity is overcome through the use of LLM Distillation. The pipeline's performance was optimized using a sequential greedy search ablation study and bench-marked against a basic, rule-based NLP baseline that utilizes SpaCy and VADER packages.

The following objectives are established to achieve the aim successfully, without ambiguity:

- **Dataset Engineering:** Create a high quality training dataset within computation constraints
- **Aspect Extraction:** Develop a token classifier (BIO tags) to accurately identify fit related clothing features
- **Sentiment Quantification:** Engineer a classification model that accurately assigns fit sentiment on an ordinal scale
- **Hyperparameter Optimization:** Maximize the systems macro F1-score using a sequential greedy search ablation study
- **Evaluation and Application:** Compare the final developed architecture against a rule based baseline and demonstrate commercial use cases.

2 Literature Review

The rise of e-commerce has led to an explosion of unstructured consumer feedback, necessitating advanced computational methods to generate actionable insights. Analyzing this data is crucial for product improvements, inventory forecasting and recommendation systems. Traditional sentiment analysis as-

signs a single polarity (e.g. positive, negative, or neutral) to an entire text. However, as identified by Shobana (2022) [5], this method fails when applied to complex reviews containing multiple, often conflicting sentiments towards different features simultaneously. A review with multiple features mentioned cannot be accurately represented by a single sentiment classification without losing information.

To resolve this limitation, NLP utilizes Aspect-Based Sentiment Analysis (ABSA). ABSA is a multi-stage process that isolates specific opinion targets (aspects) and quantifies the sentiment independently for each target. Early methods heavily depend on rule-based architectures and predefined word look-ups. Research by Singh and Verma (2020) [6] demonstrated the use of Part-of-Speech tagging and dependency parsing using libraries such as spaCy and VADER. While these pipelines provide a computationally inexpensive baseline, Chris et al. (2024) [7] highlights that rule based approaches struggle with complex syntax, sarcasm and contextual coreference.

A breakthrough in ABSA performance came from the introduction of the transformer architecture by Vaswani et al. (2017) [8]. The transformer utilizes a self-attention mechanism allowing the model to simultaneously weigh the influence of all words in a review on another. Devlin et al.(2018) [9] advanced this with Bidirectional Encoder Representations from Transformers (BERT) which uses a language modeling objective during pre-training to develop deep, contextually rich representations. For token level tagging (BIO) and classification tasks, the bert-base-uncased (huggingface.co, 2024) [4] architecture, comprising 12 hidden layers, 12 attention heads and 110 million parameters, was chosen. This model provides a balance of performance and computational feasibility. While newer models like RoBERTa could offer slight robustness improvement, BERT was selected for simplicity, operating within hardware constraints. The bidirectional nature of BERT allows for context before and after the aspect, which unidirectional models might miss. BERT’s mathematical stability relies on Layer Normalization and advanced non-linear activations. Ba, Kiros and Hinton (2016) [10] introduced Layer Normalization, normalizing inputs for each token, stopping gradients from exploding in size across layers. Furthermore, BERT’s feed-forward networks use Gaussian Error Linear Unit (GELU), developed by Hedrycks and Gimpel (2016) [11]. GELU allows for significantly smoother gradient flow compared to standard ReLUs, allowing the network to model more complex semantic relationships.

Parameter optimization during fine-tuning is done using gradient descent. Loschilov and Hutter (2017) [12] proved a critical mathematical flaw in standard ADAM optimization, forming AdamW. AdamW penalizes large weights with weight decay and gradient independently, this helps to mitigate the risk of overfitting on noisy training data.

This pipeline models apparel fit with ordinal classification. Standard sentiment analysis uses nominal classification. Misra et al. (2018) [3] emphasizes that ordinal classification is necessary to categorize the fit semantic of clothing effectively. Nominal classification does not penalize larger errors (for example, confusing ”small” with ”large” should be a greater error than confusing ”small” with ”true-to-size”). Misra et al. also observed that consumer review datasets can suffer from class imbalance (the ”J-shaped” distribution). Addressing this requires using a Macro-F1 score to ensure that minority classes are weighted equally with majority classes.

Finally, the scarcity of manually annotated training data can present a major bottleneck for ABSA tasks. Nevéditsin et al. (2025) ([13]) demonstrates that knowledge distillation architectures can resolve this

issue. This uses generalized LLMs to programmatically label a massive dataset with a strong prompt. The smaller bert-base-uncased model can then inherit the complex semantic reasoning capabilities with a fraction of the computational power. Maximizing this distilled model’s performance requires hyperparameter optimization. Chickering (2002) [14] gave mathematical proof for Greedy Search algorithms. This validates that by sequentially holding parameters constant while adjusting another (e.g. weight decay), a more optimal structure configuration can be found efficiently without brute forcing every possible combination.

3 Methodology and Theory

3.1 Aspect Extractor Forward Pass

The goal of the Aspect Extractor is to identify the particular subject (aspect) of a sentence. In the context of clothing reviews, an Aspect Extractor should identify what the customer is talking about in their review. This section will follow the extractor pipeline of the review ”This shirt fits well”, extracting the aspect ”shirt” via the mathematical procedure.

The transformer model selected for this process was bert-base-uncased, loaded via the HuggingFace transformer library. Architecturally, this model has 12 hidden layers (Transformer blocks), 12 attention heads, a vector dimension of 768 (hidden size), and approximately 110 million total parameters.

This specific configuration was chosen because its bidirectional attention mechanism provides the robust contextualization required for accurate token-level classification. The internal parameters of bert-base-uncased were pre-trained on large amounts of data to develop generalized contextual representations of the English language. For this project, a custom linear neural network head is added to the top of this model. This head is designed to perform token-level classification, translating BERT’s contextual embeddings into specific BIO (Begin, Inside, Outside) tags to successfully find the target aspects from a review.

Tokenization and Input Embeddings: The first step of this Aspect Extraction Pipeline is to convert the input sentence into tokens, a process known as tokenization. A token is a fundamental unit of text that words and sentences can be broken down into. Rare or large words are often broken down into multiple tokens. This model has a specific vocabulary of tokens that it calls upon for words and punctuation.

$$\text{This shirt fits well} \rightarrow [CLS], 'this', 'shirt', 'fits', 'well', [SEP] \quad (3.1)$$

[CLS] and [SEP] are special tokens in BERT. CLS (Classification) is placed at the beginning to absorb entire sentence context while SEP (separator) marks the end of sentences. This sentence therefore consists of 6 tokens. The AutoTokenizer module from transformers library is used for this model.

The next step, known as Vectorization, is to convert these tokens into Vector Embeddings. These initial

embeddings are mapped, from BERTs look up library, uniquely to each token. Initially, these vectors do not embed the meaning of the word in its given context. In this case, these are 768 dimensional vectors, with an example shown in Figure 3.2 .

$$\text{“Shirt”} \xrightarrow{\text{Tokenization}} \text{‘shirt’} \xrightarrow{\text{Vectorization}} \begin{bmatrix} 0.124 \\ -0.531 \\ 0.082 \\ \vdots \\ 0.892 \end{bmatrix}_{768 \times 1} \quad (3.2)$$

It is these embeddings that will adjust throughout the attention mechanism to better encapsulate (embed) the contextual meaning behind each token. Conceptually, similar words will have similar vectors, pointing in similar “directions” in this high dimensional vector space.

Each word in the input sentence is converted into its corresponding vector embedding where they are concatenated to form Matrix X, size $[n \times 768]$. n is the number of input tokens, which in this case is 6. Each row of this matrix is a token vector embedding; therefore it has 768 columns.

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,768} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,768} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,768} \end{bmatrix}_{n \times 768} \quad (3.3)$$

Multi-Head Attention Mechanism: The input to the Multi-Head Attention Mechanism is matrix $\mathbf{X}$. The first row represents the first token of the input.

To get the Query (Q), Key (K) and Value (V) matrices of size $[n \times 768]$, the following equations are utilized [8].

$$Q = XW^Q \quad (3.4)$$

$$K = XW^K \quad (3.5)$$

$$V = XW^V \quad (3.6)$$

W^Q , W^K , W^V are matrices of size $[768 \times 768]$ whose parameters have been tuned the in pre-training of BERT. These matrices are then split up into the BERT model’s 12 attention heads. The vectors are 768 dimensional and so dividing them amongst 12 attention heads yields each new matrix of size $[n \times 64]$.

The large Query, Key and Value matrices $[n \times 768]$ are partitioned into 12 smaller matrices $[n \times 64]$ for Multi-Headed Attention.

$$Q = [Q_1 | Q_2 | \dots | Q_{12}], \quad K = [K_1 | K_2 | \dots | K_{12}], \quad V = [V_1 | V_2 | \dots | V_{12}] \quad (3.7)$$

The following equations follow the pipeline of 1 of these 12 heads. First, by multiplying the first heads Query matrix [6 x 64] and transposed Key matrix [64 x 6], an Attention Score matrix is calculated of size [6 x 6].

$$\text{Attention Score}_1 = Q_1 K_1^T \quad (3.8)$$

The Attention score is then divided by the square root of the heads dimension size ($d_k = 64$) which is then passed through the Softmax function, yielding the Attention Weights [6 x 6], a matrix of percentages. Conceptually, this Attention Weights matrix shows how much each token contributes contextually to each other token.

$$\text{Attention Weights}_1 = \text{softmax} \left(\frac{Q_1 K_1^T}{\sqrt{d_k}} \right) \quad (3.9)$$

A final attention head matrix, Z_1 [6x64], is calculated by multiplying this Attention Weights matrix [6 x 6] by the heads Value matrix [6x64].

$$Z_1 = \text{softmax} \left(\frac{Q_1 K_1^T}{\sqrt{d_k}} \right) V_1 \quad (3.10)$$

Each of the 12 attention heads follows this same procedure. These (Z_{1-12}) matrices are concatenated and then multiplied by W_0 , a pre-tuned weight matrix [768 x 768]. This forms $\mathbf{Z}$, a matrix showing how the original input can be adjusted to more effectively capture how each token influences each other.

$$Z = \text{Concat}(Z_1, Z_2, \dots, Z_{12}) W^O \quad (3.11)$$

Matrix Z is added to the original input X and passed through a Normalization (LayerNorm) to form Output1.

$$\text{Output}_1 = \text{LayerNorm}(X + Z) \quad (3.12)$$

Output 1 is passed through a two-layer linear network:

$$\text{Output}_2 = \text{LayerNorm}(\text{Output}_1 + FFN(\text{Output}_1)) \quad (3.13)$$

Output_2 is the final matrix [n] x 768] representing the first hidden layers newly contextualized matrix of token vectors. This entire sequence is then repeated 12 times (one pass for each hidden layer) with Output_2 from layer 1 being passed in as an input to layer 2 replacing X in the initial input (3.3).

Layer Normalization and Feed-Forward Network: The LayerNorm function [10] is used to normalize each token vector in the matrix such that it has a mean value, μ , of 0 and variance, σ^2 , of 1 among the 768 values.

$$\text{LayerNorm}(h) = \gamma \odot \left(\frac{h - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta \quad (3.14)$$

Where h is the input vector for a single token, μ is the calculated mean of the elements within h . σ^2 is the variance of the elements within the vector. ϵ is a small constant, 1×10^{-5} added to the variance to prevent division by zero. γ and β are learned parameter vectors (same size as h) that scale and shift the normalized vectors. Without normalization, updating weights with each layer can become unstable.

The Feed Forward Network (FFN) processes the contextual information gathered by the attention layer, applying a GELU function [11] to learn more complex, non-linear relationships between the tokens.

$$\text{FFN}(X) = \text{GELU}(XW_1 + b_1)W_2 + b_2 \quad (3.15)$$

X is the input matrix, for example Output_1 from (3.12) (6×768), W_1 is the first learned weight matrix (768×3072) used to expand vector dimensions and identify more complex relationships. b_1 is the first learned bias vector (3072). GELU (Gaussian Error Linear Unit) activation function is applied to introduce non linearity. W_2 is another learned weight matrix (3072×768) used to project the expanded matrix back to the original shape, and b_2 is the final learned bias vector (768).

Classification Head and Logit Extraction: The matrix that is produced as a result of the 12 hidden layer attention mechanism represents a richer meaning behind each token in the input sentence with context of the other tokens in the sentence. To finish the Aspect Extraction Forward Pass, each newly enriched token vector is passed into a final Linear Classification Head.

$$Z_i = Wh_i + b \quad (3.16)$$

h_i represents the i_{th} token vector, each a 768 dimensional vector (768×1). W is the final Weight Matrix (3×768) and b is the final bias vector (3×1). The parameters of these matrices are found through training. Z_i is a vector (3×1) of logits representing Aspect tags, O, B-ASP or I-ASP. O (outside) represents a token that is not the Aspect, B-ASP (Beginning of Aspect) represents a token that is at the start of an Aspect and I-ASP (Inside of Aspect) represents a token that is inside of an aspect.

$$Z_i = \begin{bmatrix} O \\ B - ASP \\ I - ASP \end{bmatrix} \quad (3.17)$$

For each token, these are computed, the logit with the highest value is selected to represent that token. In the example review, "This shirt fits well", a well trained model would label the word (and token) "shirt" as B-ASP and all other tokens as O, identifying that the "shirt" is the aspect of this sentence.

In alternative situations where the probability of each BIO tag is needed, the $\text{softmax}(Z_i)$ can be used, yielding the percentage confidence the model has in each tag class.

3.2 Aspect Extractor Backward Pass (Training)

The final output of the forward pass was tensor Z_i (3.17) representing the logits of the different aspect classes: O, B-ASP and I-ASP. While the forward pass generates predictions, without training, these predictions are random since the Linear Classification head is initialized with Gaussian random weights and biases. The backward pass is the process by which the model learns and tunes its parameters. This happens in four main steps: applying the forward pass, quantifying the error (loss), calculating the gradients of each parameter, updating the 110 million parameters (optimization).

In this example, "Full Fine-Tuning" is employed since both the new classification head and the pre-trained BERT body are mathematically updated.

During the training process, the forward pass computes multiple data samples simultaneously, in the script, `PerDeviceBatchSize = 8` and so the forward pass is computed in batches of 8. This means that 8 training review sentence are processed in one iteration of the Backward Pass.

Softmax and Cross-Entropy Loss: Firstly, the raw logits (Z) from the classification head must be converted into a probability distribution. This can be performed using the Softmax Function [15], which normalizes the logits into percentages that sum to 1.

$$\hat{y}_c = \frac{e^{z_c}}{\sum_{j=1}^3 e^{z_j}} \quad (3.18)$$

Where $\hat{y}_c$ is the model's predicted probability for a specific class c (O, B-ASP, or I-ASP). z_c is the raw logit score for class c . The denominator is the sum of the exponential of all 3 class logits for that specific token, ensuring the resulting probabilities sum to 1.

These predicted probabilities ($\hat{y}$) are compared to the true label (y) provided by the training dataset. The true label has a probability of 1 assigned to the aspects and 0 to the others (e.g., if the true class is B-ASP, $y = [0, 1, 0]$). To calculate the error, the model uses the Categorical Cross-Entropy Loss function [15]:

$$L = - \sum_{c=1}^3 y_c \ln(\hat{y}_c) \quad (3.19)$$

Where L is the calculated error (loss) for a single token. y_c is the true probability of class c (which is exactly 0 or 1). $\ln(\hat{y}_c)$ is the natural logarithm of the model's predicted probability for class c .

Since y_c is 0 for all incorrect classes, this equation simplifies (Equation 3.20). If the model predicts

a 0.1 probability for the correct class, the loss is high, if it predicts 0.99, the loss is near zero.

$$L = -\ln(\hat{y}_{\text{correct}}) \quad (3.20)$$

This per token loss is computed for each token in each sentence of the batch. The mean loss across the entire batch is then calculated to represent the batch performance. This average loss is what is passed through to the Backpropagation process.

Backpropagation and Gradient Calculation: To update the weights, the model must determine exactly how much each individual parameter (of which there are 110 million) contributes to the overall loss L . This is achieved through backpropagation and utilizes the chain rule. The partial derivative of the loss with respect to the raw logits (gradient) simplifies to the difference between the prediction and the truth.

$$\frac{\partial L}{\partial z_c} = \hat{y}_c - y_c \quad (3.21)$$

Where $\frac{\partial L}{\partial z_c}$ is the gradient (slope) of the loss with respect to the logit z_c . $\hat{y}_c$ is the model's predicted probability. y_c is the true label (1 or 0).

This gradient flows backward into the classification head's weight matrix using the chain rule. The same logic applies to the bias tensors.

$$\frac{\partial L}{\partial W_{\text{head}}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial W_{\text{head}}} \quad (3.22)$$

Where $\frac{\partial L}{\partial W_{\text{head}}}$ is the resulting gradient tensor for the classification head's weight matrix. It is the same size as the weight matrix itself (3×768). The right side of the equation represents the continuous multiplication of partial derivatives via the chain rule.

Since the model undergoes Full Fine-Tuning, the chain rule does not stop at the head, it continues down deeper into the pre-trained BERT model. The following example shows how this applies to the Query matrix in the 12th Attention hidden layer:

$$\frac{\partial L}{\partial W_{\text{layer12}}^Q} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial (\text{Attention_Output})} \cdot \frac{\partial (\text{Attention_Output})}{\partial W_{\text{layer12}}^Q} \quad (3.23)$$

Where W_{layer12}^Q is the query weight matrix in the 12th transformer layer. h represents the hidden state vector outputted by the BERT model.

Modern deep learning frameworks like PyTorch use automatic differentiation to compute these chain rules efficiently. This process produces a gradient tensor of identical shape for every single one of the 110 million parameters, showing how each parameter influences the loss.

Weight Optimization with AdamW (Gradient Descent): The final step is to update the weights using the found gradient tensors. This requires optimization which is typically applied by some form of Gradient Descent, a way of stepping down towards a global minimum loss in a high dimensional weights plane. Standard Gradient Descent follows the weight update formula:

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial L}{\partial W_{\text{old}}} \quad (3.24)$$

Where W_{new} and W_{old} are the updated and current weight matrices. η is the learning rate (e.g., 2×10^{-5}), which controls the step size taken down the gradient. $\frac{\partial L}{\partial W_{\text{old}}}$ is the calculated gradient tensor.

Standard Gradient Descent assigns a uniform learning rate to all parameters, this can be inefficient and take many iteration steps to find loss minima, especially in when finding regions of low gradient.

This project utilizes a more advanced, highly efficient optimizer called AdamW (Adaptive Moment Estimation with Weight Decay) [12]. By keeping track of momentum (the running average of past gradients), Adam can take bigger steps when gradients point in the same "direction" historically, enabling it to power through small bumps in the loss landscape and find minima in less iterations. Adam also tracks variance to dynamically adjust the learning rate for every parameter individually. This accelerates convergence and prevents getting stuck in local minima.

This process also uses Weight Decay. This decay slightly shrinks the weights after each update, forcing the gradient updates (which pushes useful weights up) to fight against the decay (which pushes weights down). This prevents the model from overfitting on noisy parameters and ensures the model generalizes better to unseen reviews. This weight decay is a useful training parameter that can be adjusted within the Training Arguments function of the transformers python module. AdamW follows the following general equation:

$$W_{\text{new}} = W_{\text{old}} - \eta \left(\frac{m}{\sqrt{v} + \epsilon} \right) - (\eta \cdot \lambda \cdot W_{\text{old}}) \quad (3.25)$$

Where m is the 1st moment (momentum / running average of past gradients). v is the 2nd moment (variance / running average of past squared gradients). ϵ is a small constant added to prevent division by zero. λ is the weight decay coefficient (e.g., 0.01). The final term $(\eta \cdot \lambda \cdot W_{\text{old}})$ is the weight decay penalty being subtracted directly from the weight.

Training Loop: The forward pass processes these sentence in the batch of 8 simultaneously. The back-propagation and AdamW update happen once per batch. ,with 1500 reviews, for example, the script would compute $1500/8 \approx 188$ batches. Training the model on all the training examples constitutes one Epoch, the script then repeats this over multiple epochs, depending on the scripts input parameter. Increasing the number of epochs can reduce training error. However, too many can lead to overfitting, where the model memorizes the training data and performs poorly on new data.

Following training (Backward Pass) of the Extractor, the BERT model and classification parameters

have been fine tuned for the ABSA task. Given an input of a customer review, the model goes through the forward pass process, producing an aspect classification (O, B-ASP, I-ASP) for each input token indicating the subject of the customer sentiment.

The output aspect of the extractor must be prepared and packaged into a aspect-review pair before entering the quantifier. This is performed programmatically, concatenating the review and the aspect and tokenizing them. This part is known as the pair constructor

$$\text{This shirt fits well} + \text{Aspect} \rightarrow [CLS], 'this', 'shirt', 'fits', 'well', [SEP], 'shirt', [SEP] \quad (3.26)$$

Where between the [CLS] and first [SEP] (reading left to right) is the input sentence tokenized, and between the first and second [SEP] is the aspect token/tokens selected by the Extractor [9].

3.3 Quantifier Forward Pass

The next stage of the ABSA pipeline is the Quantifier. The Extractor outputs the subject (aspect) of the customer review, the pair constructor then combines the review and aspect. The quantifier takes this pair and categorizes the aspect-based sentiment in the review.

In this pipeline, the sentiment is categorized using ordinal classification, into 4 classes, (0,1,2,3). These classes represent the sentiment towards the "fit" of the aspect, where 0,1 and 2 represents small/tight, perfect fit, and large/loose respectively. The class, 3, has the label "irrelevant". This is a class for when the extractor yields an aspect but the customer doesn't provide sentiment towards the items fit.

The Quantifier model uses the same BERT body (before fine-tuning) as in the Extractor, using 12 hidden layers, 12 attention heads and a hidden size of 768. On top of this body, a new classification head is employed to categorize the aspects sentiment.

BERT body: The BERT body in the Quantifier follows the same forward pass procedure as in the Extractor. These input tokens (3.26) are turned into 768 dimensional vectors and passed through 12 attention hidden layers, a LayerNorm [10] and FFN function. Yielding more contextually rich vector representations of the tokens. The attention equations (Equations 3.4 to 3.14) allow the model to calculate attention weights between the review words and the aspect word across the [SEP] token.

Classification Head: In the aspect extractor, the linear classification head needs to be applied to every single token (3.16). For the quantifier, the linear classification head is only applied to the first token in the sequence: the [CLS] token. As mentioned previously, this token acts as a "sponge", absorbing the necessary context from all other tokens in the input via the attention mechanism. The head becomes:

$$Z_{[CLS]} = Wh_{[CLS]} + b = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} small/tight \\ perfect \\ large/loose \\ irrelevant \end{bmatrix} \quad (3.27)$$

Where W is a weight matrix sized $[4 \times 768]$. h_{CLS} is the context rich $[CLS]$ vector $[768 \times 1]$ following the attention mechanism. b is a bias vector $[4 \times 1]$.

The model outputs raw logits for the four fit classes. By applying the Softmax function, these logits are converted into a probability distribution. The final prediction, and its corresponding percentage is retained as the model's confidence score. This allows the system to flag predictions that may require human review if the confidence is exceptionally low.

The final output is formatted with the aspect, ordinal fit classification, and confidence score. For the "This shirt fits well" example, the entire sequence is expected to Figure 3.1.

```
"Aspect": {
  "fit": "Class",
  "confidence": Softmax probability
}

"shirt": {
  "fit": "perfect",
  "confidence": 0.992
}
```

Figure 3.1: Example output structure for aspect-based fit prediction

3.4 Quantifier Backward Pass (Training)

The quantifier undergoes a similar Backward Pass procedure, taking the Softmax of the output logits. The cross entropy loss (3.19) was previously calculated for every token in the sentence and then averaged. In the quantifier, the loss is only calculated once per sentence-aspect pair, comparing the predicted $[4 \times 1]$ probability distribution of the $[CLS]$ token against the single true label for that review-aspect pair. The test data provides the exact sentiment of the aspect, for example a sentiment of tight for a given aspect would be presented as shown in 3.28.

$$\begin{bmatrix} small/tight \\ perfect \\ large/loose \\ irrelevant \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.28)$$

This cross entropy loss is then averaged (mean) across the test batch size and undergoes the same Back-propagation and Gradient Descent procedure, tuning the 110 million parameters in the Classification Head and BERT body. Thus training the Quantifier model to identify the fit sentiment for a given aspect.

3.5 Full ABSA Pipeline Summary

The complete Aspect-Based Sentiment Analysis (ABSA) architecture operates as a sequential, two-stage pipeline, taking raw review text and transforming it into an aspect specific ordinal fit classification.

In the first stage, the raw customer review is tokenized and passed through the fine-tuned Aspect Extractor. Utilizing its classification head, this model assigns a BIO tag to every token, identifying the specific garment features (aspects) the customer is discussing.

In the second stage, the original review is paired iteratively with each extracted aspect to form a Sentence-Aspect pair (separated by the [SEP] token). This pair is fed into the Sentiment Quantifier. By isolating the [CLS] token vector embedding which has absorbed the bidirectional context between the review and the specific target aspect, the classification head computes a probability distribution over the four ordinal fit classes (Small, Perfect, Large, Irrelevant). The final output of this pipeline is a structured mapping of each extracted aspect to its predicted fit class, accompanied by a confidence score. This confidence score is derived from the Softmax probability of the highest-scoring class, representing the model’s certainty in its prediction.

This architecture ensures that a single review containing multiple, conflicting sentiments (e.g., “the waist is tight but the length is perfect”) is processed correctly, producing independent fit scores for every identified aspect. The complete data flow is illustrated in Figure 3.2.

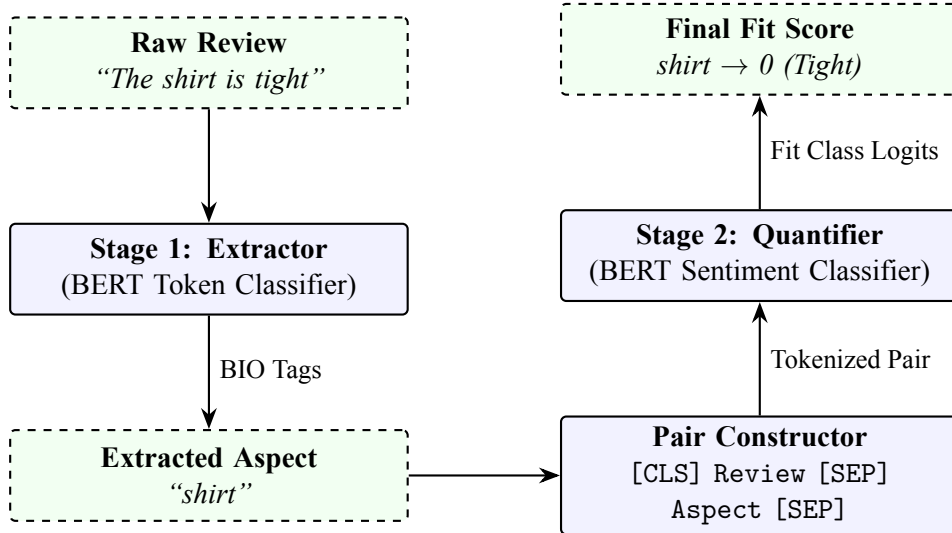


Figure 3.2: Data flow architecture of the two-stage Ordinal ABSA pipeline.

3.6 SpaCy and VADER NLP Baseline

An alternative approach to transformer-based ABSA is a rule based approach. By utilizing libraries such as spaCy and VADER (Valence Aware Dictionary and sEntiment Reasoner), a rule based ABSA pipeline can be developed. Rule-based systems are computationally inexpensive, are highly transparent and re-

quire no training data. However, they lack the contextual depth of models like BERT. They typically struggle when processing double negatives, sarcasm, spelling mistakes, slang, and complex semantic tone.

To rigorously evaluate the performance of the proposed BERT model, it is necessary to establish a performance baseline for BERT to compete against. Therefore, a rule based ABSA pipeline was developed using spaCy for aspect extraction and VADER for sentiment analysis.

Aspect Extraction (SpaCy): To identify specific clothing aspects mentioned within a review, the pipeline uses a lightweight English spaCy model [6] alongside a predefined dictionary of apparel-specific keywords (e.g., "waist", "sleeves", "collar", "fit"). Rather than simply searching the raw text for these keywords, the pipeline uses spaCy's "part-of-speech tagging" and "dependency parsing".

Firstly, the model searches for "noun chunks", phrases that have a noun as their head, allowing accurate capture of aspects even if they are multiple words like "shoulder strap". As a fallback, the process also finds individual tokens tagged by spaCy as Nouns or Proper Nouns. If a noun chunk matches a word in the curated apparel dictionary (case insensitive), it is extracted as an aspect. This allows for extraction of multiple aspects within a single review, like in the BERT pipeline.

Sentiment Analysis (VADER): Once the aspects are extracted, the pipeline must find the "fit" sentiment associated with each specific aspect. This can be difficult where multiple aspects have conflicting sentiments (e.g. "The sleeves are too tight, but the length is perfect").

To solve this, the script employs a windowing technique, splitting the review into smaller segments (windows) at conjunctions (commas, "and", "but"). For each extracted aspect, the method only processes the specific window that mentions the given aspect. Within a specific window, the window is scanned for predefined fit-related adjectives grouped into three categories: Tight/Small (e.g. "snug", "narrow"), Large/Loose (e.g. "baggy", "oversized") and Perfect/Ideal Fit (e.g. "true to size"). VADER then calculates a compound sentiment score between -1 and 1 for the window. Scores above 0.4 are deemed positive, below -0.4 are negative, and everything in between is neutral.

Classification: Finally, the pipeline combines the keywords and VADER sentiment to assign an ordinal fit classification, as was done with the BERT pipeline.

- A Small keyword paired with a negative or neutral sentiment results in a "Small" class.
- A Large keyword paired with a negative or neutral sentiment results in a "Large" class.
- A Perfect keyword paired with a positive or neutral sentiment results in a "Perfect" class.
- Any other combination is flagged as inconclusive/irrelevant.

An aspect might appear in multiple windows within a single review. Therefore, the pipeline aggregates the classifications from all relevant windows. The final label is determined by a majority vote, yielding a final ordinal classification directly comparable to the output of the BERT sequence (without the

confidence score).

4 Experimental Work and Investigation

4.1 Dataset Engineering

Transformer models like BERT require large amounts of data for effective fine tuning. Data must be task-specific and high-quality, often presenting a significant bottleneck for AI engineers. This project utilizes three main methods for Dataset Engineering:

- **Manual Annotation:** Human annotators label the data. While often yielding highest accuracy this method is highly resource-intensive and unscalable.
- **Weak Supervision:** This is where rules or heuristics are employed programmatically to automatically assign labels to data. This is highly scalable but limited by the rules in place, often creating noisy data.
- **Knowledge Distillation (LLM Labelling):** This method employs a larger, generalized LLM to annotate data, effectively "distilling" its advanced reasoning into a dataset used to train a smaller, more efficient model (like BERT). Major AI companies such as Google DeepMind [16] often distill their better, larger model (e.g Gemini 2.5 pro) to create more efficient and cheaper to run models (e.g. Gemini 2.5 Flash).

In this project, different training methods are employed. Little manual annotation is used, Weak Supervision and LLM Distillation are the main chosen methods for data labeling to balance efficiency and quality.

Source Data: The foundational data used for this project is the "Clothing Fit Dataset for Size Recommendation" [17, 18]. Specifically, the RentTheRunWay dataset was utilized, which contains 192,544 customer reviews. This dataset is highly valuable for ABSA as it contains domain specific review text alongside metadata such as item category, customer sizing, and global fit feedback. All of this is in JSON format.

Extractor Data Generation: The Aspect Extractor is a token-level classifier. To train it, the model requires data where every token is mapped to a BIO tag (Begin-Aspect, Inside-Aspect, Outside). The training of the specific model [4] requires a specific JSON format (Figure 4.1).

```
{
  "tokens": ["[CLS]", "This", "suit", "is", "very", "tight", ... "[SEP]"],
  "input_ids": [101, 1045, 12524, 2023, 4377, 2005, ... 102],
  "ner_tags": [0, 0, 1, 0, 0, 0, ... 0]
}
```

Figure 4.1: Extractor Training Format

Note: In the example above, the 3rd token ("suit") aligns with the B-ASP tag (1).

The input ID's are the tokens converted into BERT's pre defined vocabulary (a specific integer index). NER tags (Named Entity Recognition) is simply the token BIO classification. 0,1,2 map to Outside, Begin-Aspect and Inside-Aspect respectively.

Quantifier Data Generation: The sentiment Quantifier evaluates the sentiment of a specific aspect within the context of a review. Determining ordinal fit (Small/Tight, Perfect, Large/Loose, Irrelevant) is a complex semantic task that simple keyword rules struggle to capture accurately. The architectural requirement of the Quantifier JSON dataset structure can be seen in Figure 4.2.

```
{
  "text": "The waist was incredibly tight but I loved the fit of the dress
  .",
  "text_pair": "waist",
  "label": 0
},
{
  "text": "The waist was incredibly tight but I loved the fit of the dress
  .",
  "text_pair": "dress",
  "label": 1
}
```

Figure 4.2: Example of aspect-specific input formatting

The text pair is the specific aspect for the training example and the label refers to its ordinal class (0,1,2,3 are Small, Perfect, Large, Irrelevant respectively). This process must ensure that if a single review contains multiple aspects with conflicting sentiments, they are separated into distinct, independent training examples.

Dataset Filtering: Real-world datasets can be heavily imbalanced. The paper "Overcoming the J-Shaped Distribution of Product Reviews" [19] suggests product reviews follow a "J-Curve", demonstrating that the total volume of positive reviews statistically outnumber the number of negative reviews. This was validated by the RentTheRunway dataset, of the 192,544 reviews, 72% had overall sentiments of "fit". To stop the model from over-predicting the majority class, a filtering script was used to extract 1000 reviews of "Small", "Fit", and "Large" reviews, to then be used by varying data labeling techniques.

Knowledge Distillation Architecture: While commercial LLMs offer high quality data annotation, relying on cloud-based APIs for large-scale dataset generation can present significant constraints. In an early attempt to generate data, Gemini 2.5 flash API was utilized, however, rate-limiting and costs quickly proved to be a bottleneck. To overcome this, a custom local distillation pipeline was created.

Instead of relying on commercial API's, an open source model (OpenAI's "gpt-oss-20B") [20] was deployed locally. Running a 20 billion parameter model requires significant V RAM and therefore a MacBook Pro with 32GB of unified memory was employed, acting as an inference server using LM studio [21]. The primary data processing scripts were executed on a separate windows machine, utilizing the Mac over a local network, this process completely eliminated API costs related to data labeling.

This process requires strict prompt engineering to ensure the strict JSON format required by the BERT Extractor and Quantifier models. Distilling thousands of reviews is a time intensive process and therefore this pipeline was designed with several error handling techniques to avoid timeouts, repeat review labeling, and accurate responses.

To guarantee the quality and consistency of distilled data, it is vital to perform rigorous prompt engineering. high performing generalized LLM's can be prone to hallucination, inconsistent formatting and conversational filler. Therefore, the inference prompts are structured with strict instruction's and constraints.

Extractor Constraints: The extractor LLM distillation was governed by the following extraction rules:

- **Strict Sentiment Criteria:** The model was commanded to extract an aspect only if its fit was explicitly described as a synonym of "Small/Tight", "Perfect/ Fit well", or "Large/Loose". Ambiguous statements (e.g. "weird fit") or general unrelated sentiments were strictly ignored, reducing noise in the training data.
- **Contextual coreference resolution:** In natural language, it is common for subjects to be replaced with pronouns, the model is instructed to resolve pronouns to their specific garment target if there is one, this process is known as contextual coreference resolution. For example, if the review is "I bought a dress and it was too big" the model is forced to extract "dress" rather than the pronoun "it".
- **Phrase Preservation and Granularity:** The model is required to identify highly specific garment parts whilst also strictly preserving multi-word aspects rather than splitting them into separate aspects.
- **Deterministic Formatting:** The LLM was heavily constrained to return the exact valid JSON format, it is instructed to ignore markdown formatting and conversational padding which often occur in LLM's.

The capable 20B parameter generalized model paired with this strict prompt structure can successfully label high-fidelity training data. This ensures the resulting dataset accurately represents the nuanced classification required to train the BERT Extractor.

4.2 Evaluation Metrics

To rigorously assess the performance of different BERT models and the rule based baseline (SpaCy and VADER), objective performance metrics are required. The ABSA pipeline is divided into two distinct tasks, Aspect Extraction and the ordinal Sentiment Classification (Quantifier). The metrics used to evaluate each stage must differ slightly to accurately reflect their specific objectives.

In NLP, its important to test and quantify a models ability to generalize unseen data, models can often be overfit on training data. In this project, evaluation metrics serve multiple purposes. They establish a quantitative baseline using the rule-based approach, providing targets for the BERT model to surpass.

They help identify where a model is failing, for example if the model is aggressively guessing too many aspects (low precision) or being too conservative and missing them (low recall). In real world data, classes are rarely balanced, for example there may be many more 'Irrelevant' reviews than other classes. Specific metrics like F1 scores ensure the model's performance is evaluated fairly across all classes, preventing one class from masking another classes performance.

Aspect Extractor: The extractors goal is to isolate specific aspects from a review. In this context, True Negatives are generally ignored, as counting every single word the model correctly did not extract is not helpful (the majority of words aren't aspects). Instead, performance is calculated by comparing sets of predicted aspects against ground-truth (labeled) sets.

- True Positive (TP): The model correctly extracted an aspect that was present in the ground truth (e.g. both the dataset and model identified "waist")
- False Positive (FP): The model extracted a word that was not an aspect in the ground truth (e.g. the model incorrectly extracted "the" as a clothing aspect")
- False Negative (FN): The model failed to extract an aspect that was in the ground truth (e.g. the dataset labeled "sleeves" but the model misses it)

Using these definitions, the Extractor is evaluated on three main metrics, Precision, Recall, and F1 Score, all given as a percentage or a score of 0 to 1.

$$Precision = \frac{TP}{TP + FP} \quad (4.1)$$

$$Recall = \frac{TP}{TP + FN} \quad (4.2)$$

$$F1 \text{ Score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (4.3)$$

Out of all the aspects the model guessed, the precision shows what percentage of them were actually correct. With a high precision, the model is trustworthy when it makes a prediction.

Out of all the true aspects in the dataset, the recall shows the percentage of these the model successfully finds. If the recall is a high percentage, the model is thorough and rarely misses aspects.

The F1 score is a popular metric when evaluating models, this provides a balanced metric that penalizes models that favor recall over precision or vice versa. This is the harmonic mean of Precision and Recall, a score of 0.9 or 90% would be a high performing model.

Sentiment Quantifier (Ordinal Classification): The Quantifier has to categories aspects into four classes: 0 (Small/Tight), 1(Perfect Fit), 2 (Large/Loose) or 3(Irrelevant). Since there are multiple classes, TP, FP, FN and TN are calculated relative to each specific class (e.g. evaluating the performance metrics of class 0 specifically).

- True Positive (TP): The system correctly predicts the specific class (e.g. the dataset label is Loose/Large and the model predicts Loose/Large)
- False Positive (FP): The pipeline incorrectly predicts this class when it should have been another

- False Negative (FN): The system fails to predict this class when it was actually the correct class
- True Negative (TN): The system correctly finds that the aspect does not belong to this specific class (e.g. the true label is Small/Tight and the system correctly does not guess Large/Loose).

These can be used to calculate the same Precision, Recall and F1 scores per class (equations 4.1,4.2,4.3). To evaluate the Quantifier performance as a whole, the following tools are employed:

- The Confusion matrix: The specific confusion matrix for this quantifier is of size 4×4 , this visualizes the model's predictions against true labels.

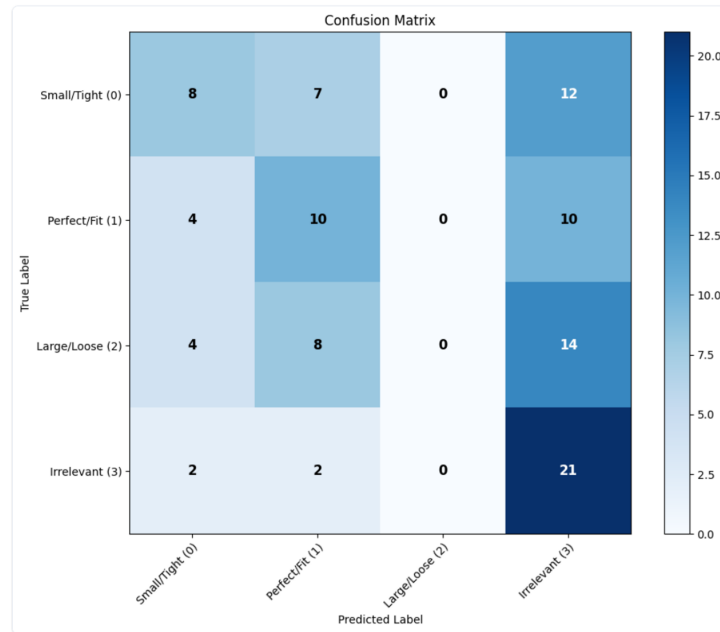


Figure 4.3: Early model design confusion matrix

The diagonal (top left to bottom right) represents True Positives. This is particularly useful because it shows exactly how the model is failing. For instance, Figure 4.3 shows an early model design confusion matrix where it clearly shows that this model never predicted Large/Loose, indicating poor model performance and a preference to predict "Irrelevant".

- Macro Average F1 Score: The dataset may inherently have more of one class than another and so standard accuracy results can be misleading. A model that blindly guesses "Perfect" every time might achieve high accuracy simple because the dataset has a large proportion of "Perfect" fit reviews. To counter this, the pipeline calculates the Precision, Recall, and F1-score for each individual class independently, and then calculates the Macro F1-Score (unweighted mean of the four F1-scores). This Macro F1-score ensures that the models ability to identify a smaller category is treated as equally important as identifying more common classes. Thus providing a robust metric to analyze the Quantifiers performance.

4.3 Experimental Setup and Ablation Study

The objective of this ABSA pipeline is to design, train, and evaluate a high-performing, transformer-based ABSA pipeline. This is defined by the model’s ability to accurately find specific garment aspects and successfully categorize their ordinal fit when met with new data.

The performance goal for the final pipeline is to convincingly outperform the rule-based baseline such that the increase in computational cost is justified. To achieve this, the experimental phase is structured into a systematic ablation study, investigating the performance of the system by isolating, removing, or altering specific parameters to understand how they contribute to the overall system performance. The parameters will be tested independently to show clear cause-and-effect relationships. This ablation study will be performed for both the Extractor and the Quantifier.

Each system tested will be evaluated with an 80%-20% split of training to test data. These models will undergo evaluation using the established performance metrics, Precision, Recall, and F1 score, alongside a confusion matrix where applicable to diagnose specific class failures.

Parameters: In deep learning, hyper-parameters mathematically interact (e.g. altering the batch size changes the number of gradient descent steps per epoch, which indirectly affects the optimal learning rate and weight decay). Testing every possible combination requires huge computation resources. Therefore, this study adopts a sequential greedy search strategy ([14]), parameters will be altered and tested independently against the model baseline. The highest performing parameters (highest F1 score) will then be combined into a final model design. By finding the locally optimum F1 score for each parameter, on combining, a more global optimum is found.

This study will manipulate the following variables:

- **Training Data Volume and Quality:** Weak supervision, manual annotation and LLM distillation will be utilized for labeling data. By increasing the volume of training data, or introducing manually corrected datasets, the model is expected to improve generalization, reduce overfitting, and improve accuracy, therefore leading to higher F1 scores. However, increasing training data exponentially increases training time.
- **Epochs:** This dictates how many times the model sees the entire training dataset. Too few epochs can lead to underfitting (the model fails to learn the patterns). Too many epochs can lead to overfitting (the model memorizes the training data but fails on the unseen test reviews). The goal is to find the number of epochs where the test performance stops improving.
- **Batch Size:** The number of sentences processed before a weight update occurs. A larger batch size smooths out gradient updates, reducing variance (noise) due to averaging the gradient over more samples. A larger batch size would also cause training to process epochs faster, however this requires significantly more GPU VRAM and can sometimes cause the model to converge more sharply towards a suboptimal local minima. This training is performed on an RTX 2060 with 6GB GDDR6 VRAM.
- **Weight Decay:** Weight decay penalizes large weights. Increasing weight decay forces the model to remain mathematically simple, which is expected to improve its generalization ability to unseen

reviews. However, excessive decay can prevent the models ability to learn, particularly on small datasets because there is less data to provide the strong gradients to "fight back" against the weight decay.

Table 4.1 outlines the specific parameter configurations that will be tested during this ablation study, shifting one parameter at a time from the v1 Baseline (where v1 baseline consists of 150 LLM Distilled training data, 3 Epochs, Batch Size of 8, and a Weight Decay of 0.01).

Each performance check will be performed 3 times and averaged (mean) to account for the statistical variance in neural networks. The highest performing model from each parameter will be combined to train the next models until each parameter has been iterated through. Table 4.1 shows that for example, v1.1 will be the v1 model combined with the best performing(highest F1 score) Data Volume (This is expected to be the largest data volume). Once each parameter has been varied and tested a v1.4 model will be produced, optimized specifically to overcome the computational and semantic challenges of garment fit classification. The last parameter, data method, will be tested with this refined model to see which data method performs highest on 200 unseen reviews, yielding the final v2 model.

This Sequential Greedy Search process will be performed for both the extractor and the quantifier. The architecture of the base model (12 hidden layers, 12 attention heads) is not altered for this study.

Stage	Parameter Adjusted	Input Baseline	Optimized Output
1	Data Volume	v1 Baseline	v1.1
2	Epochs	v1.1 Optimized Model	v1.2
3	Batch Size	v1.2 Optimized Model	v1.3
4	Weight Decay	v1.3 Optimized Model	v1.4
5	Data Annotation Method	v1.4 Optimized Model	v2 (Final Optimized Model)

Table 4.1: Process flow of the Sequential Greedy Search Ablation Study.

Final Pipeline Evaluation: Following the independent optimization of the Extractor and Quantifier models (yielding a v2 for both), these components will be joined together into a final complete ABSA pipeline. This final ABSA architecture will be tested against a set of manually verified unseen reviews. Its performance will be directly compared against the rule based baseline using strict Precision, Recall and F1-scores, alongside a qualitative error analysis to determine where the model fails linguistically.

5 Presentation of Results

This chapter details the evaluation of the ABSA pipeline. The performance is evaluated for the Aspect Extraction and Sentiment Quantification models independently, and then assessed as a whole pipeline.

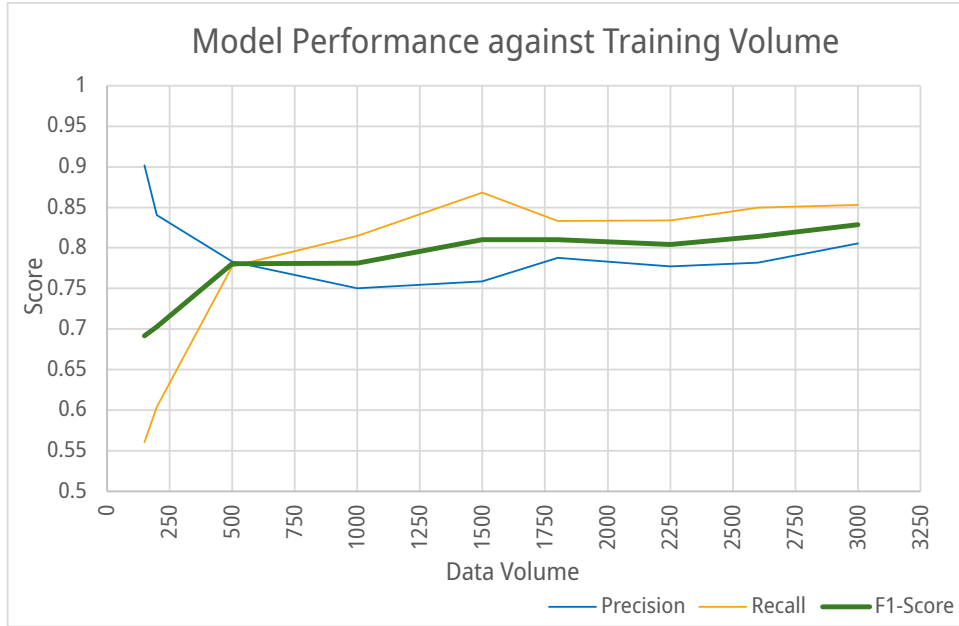


Figure 5.1: Graph to show effect of Increasing Training Volume on Model Performance

5.1 Aspect Extractor Evaluation

The objective of the aspect extractor is to accurately identify clothing features that correspond to a customers fit sentiment. Following the experimental methodology (a Greedy Search Ablation study), the parameters of the BERT model were optimized sequentially.

Training Data Volume: The initial model was evaluated by adjusting the training data size from 150 to 3000 examples (whilst keeping other parameters constant: 3 Epochs, Batch Size = 8, Weight Decay = 0.01, Data Method of LLM Distillation).

With the initial 150-data point test, the model exhibited an exceptionally high Precision (0.902) but a much lower Recall (0.561) yielding an F1-score of 0.692 averaged over 3 tests. This indicates that the model was highly conservative, the model only learned the most frequent aspects (e.g. "waist", "dress") and rarely guessed incorrectly (minimizing False Positives). This resulted in the model failing to identify less common clothing features, yielding a high number of False Negatives. As the training data volume increases to 3000 examples, the model was able to generalize more effectively and performed with progressively higher F1-scores (Figure 5.1). At 3000 data points, the model performed at its peak, with a Recall of 0.853, a Precision of 0.806 and an F1 score of 0.829 (3 d.p), +19.8% from the initial 150 data point model. Training times at this volume were still within a reasonable timeframe, with a training runtime of no more than 250 seconds. Therefore, this 3000-example model configuration became the v1.1 baseline.

Epochs: Using v1.1, performance was evaluated across 1,3,4 and 6 epochs. Results indicated that performance peaked at 3 epochs (F1: 0.829). Training for only 1 epoch (F1: 0.802) resulted in slight underfitting, while 4 epochs (F1: 0.799) and 6 epochs (F1: 0.801) resulted in performance loss and increased compute time. This decline is likely due to model overfitting where it memorizes the training

data at the expense of its ability to generalize to the 20% unseen test split. Therefore, 3 epochs were retained, yielding the v1.2 baseline.

Batch Size and GPU Optimization: Batch sizes of 4, 8, 16 and 32 were evaluated on v1.2, tracking both the F1-score and physical training runtime. Adjusting the batch size to 4 yielded the highest performance (F1: 0.833), +0.4% more than v1.2 baseline. A smaller batch size provides more frequent gradient updates per epoch and introduces slight statistical noise into the learning path, this often prevents the model from settling into suboptimal local minima.

Adjusting the batch size revealed a significant hardware bottleneck. Training was performed on an NVIDIA RTX 2060 with 6GB of GDDR6 VRAM. While batch sizes 4 and 6 processed relatively quickly (≈ 220 s), a batch size of 32 resulted in a drastic runtime increase up to 1656 seconds. A batch size of 32 likely exceeds the VRAM capacity of the GPU, forcing the system to utilize memory paging, which no longer benefits from GPU parallelization. In addition, batch sizes of 16 and 32 also performed lower on F1 score, likely tending to settle into suboptimal local minima. Therefore, v1.3 progressed with a batch size of 4.

Batch Size	Precision	Recall	F1-Score	Runtime (s)
4	0.810	0.857	0.833	224.95
8	0.806	0.853	0.829	214.85
16	0.794	0.854	0.823	326.43
32	0.791	0.825	0.807	1655.99

Table 5.1: Model performance across different batch sizes

Weight Decay: Finally, weight decay was tested across values of 0, 0.01, 0.05, 0.1 and 0.15. A high decay rate of 0.1 yielded the highest F1-score (0.841). This confirms the mathematical hypotheses that by heavily penalizing large weights (a larger weight decay), the model is forced to learn broader semantic rules rather than relying on specific noisy training data, improving the models generalization. Increasing the weight decay further to 0.15 resulted in a decline in F1 score (0.827). This large weight decay hindered the models ability to learn, particularly because this dataset (3000) is still relatively small. Therefore, a weight decay of 0.1 is selected for the optimized v1.4 model, improving on v1.3 by 0.96%.

Data Methodology Comparison: Having established the optimal hyper-parameters, the v1.4x architecture was evaluated with alternative data generation methods and models to establish the final model.

Methodology	Precision	Recall	F1-Score
SpaCy/VADER (Rule-Based)	0.252	0.579	0.351
BERT (Weak Supervision)	0.286	0.618	0.391
BERT (LLM Distillation + Manual)	0.806	0.875	0.839
BERT (LLM Distillation Only) - v2	0.808	0.876	0.841

Table 5.2: Final Extractor performance by data generation methodology.

As shown in Table 5.2, both the rule-based baseline and Weak Supervision performed exceptionally

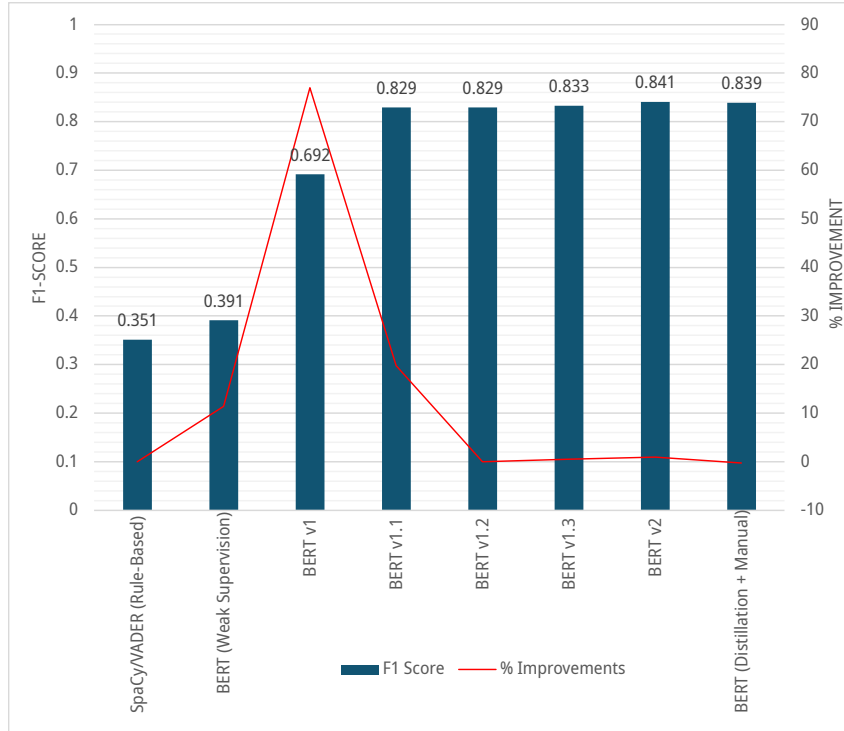


Figure 5.2: Chart to show Extractor improvements in F1-Score from previous model

poorly for this Aspect Extractor task. Notably, combining LLM distillation with 100 manually annotated corrections provided no statistically significant improvement over pure LLM Distillation (0.839 vs 0.841, well within the margin of stochastic variance typical of neural networks). This demonstrates that the engineered LLM distillation data was robust, making manual data labeling redundant in this context. Therefore the pure LLM-distilled model was finalized as the optimal v2 Extractor model, with a Data volume of 3000, 3 epochs, a batch size of 4 and a weight decay of 0.1. This final v2 model (F1: 0.841) improves on the rule based extractor $\approx 140\%$. Figure 5.2 shows the model performance progression from Rule Based to BERT v2. This clearly visualizes that the step from BERT (Weak Supervision) to BERT v1 (utilising LLM Distillation for training data) was the largest performance jump (+76.98%) of any iteration.

Failure of Keyword Spotting: While quantitative results define the overall capability of a model, it is important to examine the specific semantic limitations of the models. Both the rule-based and Weak Supervised BERT pipelines failed primarily due to a lack of contextual awareness. They acted primarily as "keyword spotters". For example, given the review: "Loved this dress, wish I could buy it!" Both models naively extracted "dress" as an aspect (yielding 1 False Positive, 0 True Positives). While "dress" is a valid keyword, the review contains no explicit fit related sentiment, violating the Extractor task. Similarly, in the review "The skirt area seemed a bit tight... I liked the dress. The material is thick" the rule based pipeline extracted "skirt", "dress" and "material", yielding 2 False Positives. The optimized v2 BERT model, which utilizes Bidirectional attention, correctly recognizes the lack of fit-related sentiment for this review and appropriately ignored the irrelevant nouns.

Ambiguity and Coreference (Extractor v2 model failure): The optimized v2 model predicted some False Positives, though often due to linguistic ambiguity. Given the review "It fit big in the back." The model extracted both "back" and "it". The ground truth only labels "back" (yielding 1 True Positive and

1 False Positive). The model identified "it" was a subject of the phrase "big" whilst in this situation, identifying "back" was sufficient to match the labeled ground truth. This False Positive highlights that NLP evaluation can often stem from subjective human labeling criteria regarding pronoun resolution, rather than architectural failures in a model.

In the event of an aspect being incorrectly extracted (False Positive), the subsequent sentiment quantifier encompasses a fail-safe with the ability to categorize a review as irrelevant where fit sentiment is not detected.

5.2 Sentiment Quantifier Evaluation

The Sentiment Quantifier performs the complex semantic task of classifying ordinal fit associated with a given review. To optimize this model, the same Sequential Greedy Search methodology was applied that was used for the Extractor. The ablation study found an optimized v2 Quantifier utilizing 3000 LLM-distilled training data points, 8 epochs, a batch size of 4, and a weight decay of 0.1.

With these hyperparameters, the v2 Quantifier was evaluated against a test dataset, achieving a Macro F1 score of 0.8361 (Table 5.3). With a Macro Precision and Recall of 0.8388 and 0.8350 respectively, exceeding the requirements.

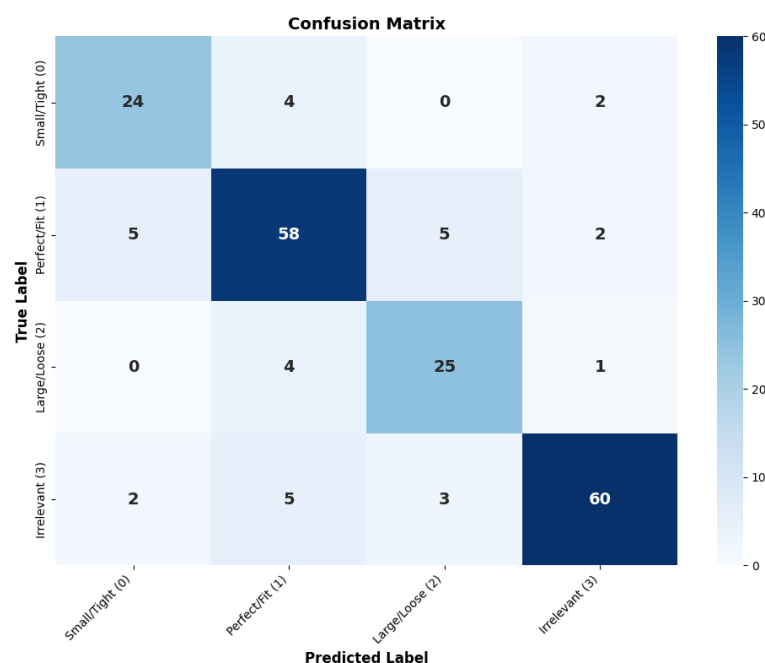


Figure 5.3: v2 Sentiment Quantifier Confusion Matrix

Class	Small/Tight	Perfect/Fit	Large/Loose	Irrelevant	Macro
F1 Score	0.7869	0.8227	0.7937	0.8889	0.8361

Table 5.3: F1 scores across different fit categories

J-Curve and Class Imbalance: Figure 5.3 shows a larger number of perfect reviews (positive) than Large or Tight (negative) with a support of 70. This falls in line with the J-Curve [19], suggesting e-commerce data inherently skews positive. Since the training data followed a similarly balanced percentage, a filtration script was implemented to represent each class evenly. Interestingly, the "Perfect/Fit" class achieved a higher F1 score (0.8227) than the "Small/Tight" (0.7869) and "Large/Loose" (0.7937) classes. Suggesting the "Perfect/Fit" reviews were less ambiguous than the other fit classes. The "Irrelevant" class performed best overall, achieving an F1-score of 0.8889. This high performance can be attributed to semantic complexity. It is simpler for the BERT model to detect a total lack of fit-related sentiment than it is to consider the nuanced ordinal fit classification. Therefore, the model excels at identifying when an aspect is purely descriptive.

The Rule Based Quantifier was independently evaluated to have a Macro F1 score of 0.3192, confirming a 161.9% performance increase when using BERT Quantifier v2. The macro F1 metric was specifically chosen to counteract any class imbalances that escaped the filtering method, ensuring minority classes are weighted equally.

Additionally, analysis of the 200-review test revealed that incorrectly classified examples were associated with lower confidence scores (mean confidence 0.61) and correct predictions were associated with higher confidence (mean 0.91). This suggests that the model's confidence score is a reliable uncertainty indicator that could be used as a flag for human review in commercial deployment.

Misclassifications: A significant success visualized in Figure 5.3 is the complete lack of misclassifications of the opposite sentiments. At no point does the model predict "Small/Tight" when the true label is "Large/Loose" or vice versa (shown by the 0 values in the confusion matrix). In a real-world application, misclassify a tight garment as loose would be a catastrophic recommendation error that could actively mislead a consumer. This outcome proves that the v2 Sentiment Quantifier model has successfully learned the semantic boundaries between the polar opposite sentiments, ensuring robust reliability. This v2 quantifier performs significantly better (+165.4%) than the early v1 model (F1 score 0.3150), previously shown in Figure 4.3, which never guessed the "Large/Loose" category.

5.3 Full ABSA Pipeline Evaluation

With both the Extractor and Quantifier models optimized, the complete ABSA pipeline was evaluated end-to-end and compared against the SpaCy and VADER rule-based pipeline. While computationally lightweight, the rule based pipeline lacks the attention required to understand the rich contextual relationship between words.

Qualitative Error Analysis: The BERT pipeline is most evidently superior in complex sentence structures involving conjunctions and conflicting sentiments. Consider the following unseen test review:

"The waist wasn't tight at all, but unfortunately, the shoulder straps were absolutely huge"

The rule based pipeline extracts "waist" and "shoulder straps". However, VADER evaluates the sentiment within a window based on word polarities. For "waist" it, detected the keyword "tight" and

therefore incorrectly assigned the aspect "waist" with a sentiment "Small/Tight". It failed to process the "wasn't" accurately, which should be interpreted as an unclear sentiment and therefore the "waist" should not even have been extracted.

The custom BERT pipeline successfully extracts only the "shoulder straps" aspect, correctly interpreting via attention that "waist" has inconclusive fit related sentiment. The pipeline also correctly finds this "shoulder strap" sentiment to be "Large/Loose" with a confidence of 94.1%. When testing the same review again by giving the Quantifier "waist" as an aspect, the Quantifier successfully identifies its sentiment as "Irrelevant" with a confidence of 83.9%, proving that the fail-safe has worked in this case. For misclassifications, the model consistently outputted lower confidence scores (between 0.45 and 0.6). Further development could implement a confidence score flag (e.g. <0.65) to indicate a human should intervene with labeling this data point.

To evaluate the pipeline as a cohesive unit, a "Strict Matching" criterion was applied to the 200 review test set. A True Positive is only recorded if the system correctly extracts the exact aspect and ordinal sentiment pair. This therefore introduced error propagation. If the extractor misses an aspect (False Negative), the pipeline automatically fails. Consequently, the performance of the end-to-end system is lower than the performance (F1 score) of its parts. In reality, the pipeline is more robust than this since the Quantifier has a built in fail-safe class ("irrelevant"). When evaluated under these strict conditions, the limitations of a rule-based approach becomes clear, while the optimized BERT architecture demonstrates clear superiority.

Pipeline Architecture	End-to-End F1-Score
SpaCy + VADER (Rule-Based)	0.248
Optimized BERT (v2 Extractor + v2 Quantifier)	0.704

Table 5.4: End-to-end performance comparison.

As shown in Table 5.4, the rule-based architecture achieved an F1-Score of just 0.248. Its Extractor stage was inherently weak (F1: 0.351), this leads to its quantifier being given inaccurate inputs, causing propagated error through its system.

Conversely, the BERT pipeline achieved an F1 score of 0.704. While this clearly shows a mathematical drop from its Extractor (0.841) and Quantifier (0.836) scores, it remains an acceptable score for this strict-match criteria. The BERT pipeline outperforms the basic rule-based pipeline by 183.9%, clearly validating that ABSA in customer reviews requires deep semantic contextualization that this Attention based architecture can provide.

6 Discussion and Application

The results demonstrate the mathematical superior performance of the transformer based architecture, however, the value of an Aspect-Based Sentiment Analysis (ABSA) pipeline comes from its commercial and consumer applications. By translating qualitative, unstructured text into quantitative data, this

system can provide insight into the e-commerce apparel sector.

For retailers, processing thousands of reviews manually is inefficient and unscalable. The proposed ABSA pipeline allows businesses to perform bulk analytics on their products. By aggregating the Quantifier’s outputs, a retailer can calculate a mean sentiment score for a specific garment aspect (ignoring irrelevant reviews). For example, if ”sleeves” are consistently classified as 2 (”Large/Loose”), this could warrant product redesign. In addition, this data can drive dynamic sizing dashboards on e-commerce websites, providing customers with guidance on how an item fits such as ”Customers report this item is true to size across the chest”.

From a consumer perspective, this pipeline can power advanced recommendation engines. To demonstrate this capability, a Proof of Concept browser extension was developed during this project using a MERN stack. By integrating the ABSA pipeline in to a chrome extension, customer reviews are located, passing them through the BERT Extractor and Quantifier, to generate insights that can be aggregated with user measurements and product size guide data to provide a recommendation for a user, overlaid on the user’s screen (Figure 6.1). This empowers consumers to make highly informed purchasing decisions without manually reading hundreds of reviews.

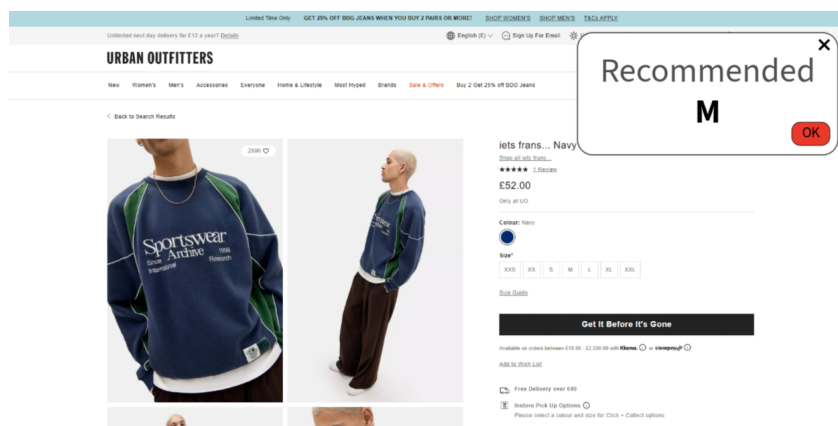


Figure 6.1: Browser Extension demonstration: Utilizing ABSA pipeline to contribute to Sizing Recommendation

6.1 Economic and Environmental Sustainability

The most significant real world impact of this pipeline is its contribution to sustainability. ”The average return rate of online apparel orders in the US is 24.4%” [1]. The high return rates are largely driven by poor fit and uncertainty. This encourages behavior known as ”bracket buying” ([2]), where consumers buy the same garment in multiple sizes with the intention of returning the ones that do not fit. This causes immense economic and logistical costs for retailers. Furthermore, the environmental cost of return shipping, increased carbon emissions, and the land filling of unsellable returned inventory is a severe sustainability issue. By providing users with highly accurate fit data before purchase, the ABSA pipeline can be utilized to prevent bracket buying, aligning e-commerce operations with broader environmental sustainability goals.

6.2 Discussion of Findings

The primary objective of this project was to design, optimize and evaluate a transformer-based ABSA pipeline capable of locating garment aspects and classifying their ordinal fit. The experimental results validate that the proposed BERT architecture significantly outperformed a traditional rule based NLP pipeline.

The overall F1 score of 0.704 achieved by the optimized BERT model represents a significant improvement over the 0.248 scored by the SpaCy and VADER pipeline. This shows a clear limitation of rule based NLP. The use of basic dictionaries render them vulnerable to complex semantics. In contrast, the attention mechanism in BERT can be used to successfully provide high dimensional representations of tokens to embed a richer semantic and contextualized meaning behind them. This allows for more accurate aspect level sentiment classification.

Furthermore, the sequential Greedy search ablation study provided insights into the specific BERT model tuning. It was observed that while scaling the volume of LLM-distilled training data from 150-3000 examples drastically improved the F1 (0.692 to 0.829, +19.8%), a larger performance increase (+76.98%F1) came from adjusting the data labeling process from Weak Supervision to LLM-Distillation (0.391 to 0.692), underlying the importance of well labeled data in the training of a BERT model and classification head, this validates Neveditsin et al. (2025) [13]. As predicted in Loshchilov and Hutter (2017) [12], increasing the weight decay in both Extractor and Quantifier improved the F1 score, forcing the network to learn more generalized semantic rules rather than memorizing the training data.

6.3 System Limitations

It is essential to acknowledge the limitations of this study. Firstly, there is domain bias. The pipeline was fine-tuned exclusively on the *RentTheRunway* dataset [17], which consists entirely of women’s rental apparel. Consequently, the model’s attention weights are trained for this demographic. This model could experience performance drop-offs if applied to menswear, children’s clothing, or footwear without further fine-tuning.

Additionally, the SpaCy and VADER pipeline used for comparison represents a relatively simple, deterministic approach. Comparing the BERT models against more advanced rule based pipelines or alternative architectures would provide a more rigorous benchmark and could be conducted in future research.

Finally, this project was under significant Hardware constraints. Training parameters were constrained by a 6GB VRAM limit, this directly affects batch size performance. To avoid commercial API costs and rate limits related to LLM distillation, a 20B parameter open-source model was hosted locally [20]. While successful, a larger volume of training data (e.g. 50,000 + examples) and a higher performing generalized LLM could potentially yield even higher performance.

6.4 Future Research

To expand on further findings of this project, future research could be conducted on a few areas. Firstly, the current Extractor identifies highly specific aspects (e.g. "collar", "neckline"). Future pipelines should implement "Aspect Clustering", grouping together related aspects under parent categories. This allows for combining synonym aspects and for more macro commercial analytics.

The current system uses bert-base-uncased [22]. Future studies should investigate the performance gains of employing stronger models, such as RoBERTa or alternative domain specific transformers. In addition, further model optimization using techniques like quantization would reduce the models required memory, lowering latency related to the models use cases (such as the browser extension backend).

6.5 Conclusion

This research successfully engineered and validated a two stage ABSA pipeline using the bert-base-uncased transformer architecture. The core objectives were to accurately quantify ordinal apparel fit sentiment and outperform the rule based baseline. The results clearly fulfill this criteria. It is demonstrated that bidirectional attention mechanisms can resolve the limitations of systems such as SpaCy and VADER. Optimizing hyperparameters through a sequential greedy search ablation study created a final BERT pipeline that achieved an end-to-end strict F1 score of 0.704, outperforming the rule-based alternative by 183.9%. Crucially, the Sentiment Quantifier showed no catastrophic miscalculations between opposite fit semantics (e.g. confusing "Small" with "Loose"), proving its reliability.

This project successfully demonstrates that using a locally hosted 20 billion parameter LLM for knowledge distillation massively reduces the need for manual data labeling and can be used as an effective alternative in low-resource situations. This approach successfully generated high quality, domain specific training data without the financial constraints of rate limits or commercial APIs.

The integration of this pipeline into a browser extension shows a clear path to future commercial use, allowing consumers to make informed purchasing decisions, directly reducing "bracket buying". This research contributes to product development, providing insights through bulk customer review processing, whilst yielding higher F1 than rule-based solutions. Environmental sustainability is improved, reducing emissions related to e-commerce returns. Future research should build on this architecture, implementing aspect clustering to provide more macro-analysis, exploring alternative transformer architectures with larger compute availability, and implementing model quantization to reduce inference latency.

7 Project Management, Sustainability, and Risk

An objective driven approach was employed for this project. A Gantt chart (Figure 7.1) was used to allocate time for data generation, model training and all other phases. This was vital due to the time intensive

nature of a ML project. This had to be adjust to account for longer training times when LLM Distillation was switched to local. In hindsight, beginning this labeling process earlier would have allowed for a larger training set, potentially improving models further. Version control was maintained using GitHub for all scripts ensuring code wasn't lost, mitigating risk associated with data loss. In addition, hardware was monitored and scripts were adapted to prevent crashing and memory overflow. While the risk of this software project is low, there is a commercial risk that the BERT pipeline could output incorrect fit classifications, potentially misleading a consumer into purchasing the wrong size. This would be largely mitigated through confidence score thresholds however the residual risk of a model confidently generating a false result remains a limitation and risk of the system.

	Date (D-MM)																											
Task	3-11	10-11	17-11	24-11	1-12	8-12	15-12	22-12	29-12	5-01	12-01	19-01	26-01	2-02	9-02	16-02	23-02	2-03	9-03	16-03	23-03	30-03	6-04	13-04				
Background & NLP Courses	X	X	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
Literature & Math Review	-	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
Rule-Based Baseline	-	-	-	X	X	X	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
Dataset Engineering	-	-	-	-	-	X	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
Train Aspect Extractor	-	-	-	-	-	-	-	-	-	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-	-			
Train Sentiment Quantifier	-	-	-	-	-	-	-	-	-	-	-	-	X	X	X	X	X	-	-	-	-	-	-	-	-			
Greedy Search Ablation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X	X	X	X	-	-	-	-	-			
MERN System Integration	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X	X	X	-	-	-	-			
Final Pipeline Evaluation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X	X	X	-	-			
Write-Up	-	-	-	-	-	-	-	-	-	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			

Figure 7.1: Gantt Chart

Sustainability is central to the ABSA pipeline, it is directly aimed at reducing "bracket buying", thus reducing returns. This leads to a reduction in carbon emissions related with return shipping and decreases the volume of unsellable returns that would otherwise be sent to landfills. Furthermore, using LLM Distillation is a sustainable AI practice, by transferring the capabilities of a 20 billion parameter into a smaller BERT model, less energy is required for inference, making it more commercially viable.

The *RentTheRunway* dataset [17], consists of anonymous customer feedback. This ensures that user privacy is preserved, following standard data protection guidelines. This project development relied on open source libraries such as spaCy [6] and Hugging Face Transformers [22]. The specific configuration, hyperparameter combination, and application into a browser extension are novel, and therefore have potential for IP protection.

The ergonomics of the desks used was considered for health and safety. Regular screen breaks were used to prevent eye strain. In addition, hardware temperatures, specifically the RTX 2060 GPU during model training and the MacBook Pro during intensive LLM distillation, were actively monitored with thermal regulation software to prevent overheating. This ensures safe operation, mitigating fire risks, and maintains the hardware in healthy condition.

All Code for this project is available on GitHub [here](#).

References

- [1] S. Zheng. The true cost of apparel returns: Alarming return rates require loss-minimization solutions, 2023.
- [2] A. Balaram, O. Perdikaki, and M.R. Galbreth. Bracketing of purchases to manage size uncertainty: Should online retailers be worried? *Naval Research Logistics (NRL)*, 2022.
- [3] R. Misra, M. Wan, and J. McAuley. Decomposing fit semantics for product size recommendation in metric spaces. In *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018.
- [4] Hugging Face. google-bert/bert-base-uncased, 2024.
- [5] G. Shobana, K.R. Baskaran, and D. Yamunathangam. Aspect based sentiment analysis for customer reviews. 2022.
- [6] A.K. Singh and A. Verma. An efficient method for aspect based sentiment analysis using spacy and vader. In *IEEE Xplore*, 2021.
- [7] E. Chris, J. Olusegun, B. Favor, and T. Ramon. Aspect-based sentiment analysis of customer reviews in e-commerce, 2024.
- [8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Cornell University (arXiv)*, 2017.
- [9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv*, 2018.
- [10] J.L. Ba, J.R. Kiros, and G.E. Hinton. Layer normalization. *arXiv:1607.06450 [cs, stat]*, 2016.
- [11] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv:1606.08415 [cs]*, 2020.
- [12] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv.org*, 2017.
- [13] N. Neveditsin, Pawan Lingras, and V. Mago. From annotation to adaptation: Metrics, synthetic data, and aspect extraction for aspect-based sentiment... *ResearchGate*, 2025.
- [14] David Maxwell Chickering. Optimal structure identification with greedy search. Technical report, Microsoft Research, 2002.
- [15] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Deeplearningbook.org, 2016.
- [16] Google DeepMind. Google deepmind, 2024.
- [17] Rishabh Misra, Mengting Wan, and Julian McAuley. Decomposing fit semantics for product size recommendation in metric spaces. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 422–426, 2018.
- [18] Rishabh Misra and Jigyasa Grover. *Sculpting Data for ML: The first act of Machine Learning*. 2021.
- [19] Nan Hu, Jie Zhang, and Paul A. Pavlou. Overcoming the j-shaped distribution of product reviews. 2006.
- [20] OpenAI, S. Agarwal, L. Ahmad, J. Ai, S. Altman, A. Applebaum, E. Arbus, R.K. Arora, Y. Bai, B. Baker, H. Bao, B. Barak, A. Bennett, T. Bertao, N. Brett, E. Brevdo, G. Brockman, S. Bubeck, and C. Chang. gpt-oss-120b and gpt-oss-20b model card. *arXiv.org*, 2025.
- [21] lmstudio.ai. Lm studio - discover and run local llms, n.d.
- [22] Hugging Face. Hugging face – on a mission to solve nlp, one commit at a time, 2024.