Natural language classification for common business use cases

Will Ashford
TrueState

6 October 2026

Abstract

Natural language classification is the problem of assigning text to a category or decision expressed in ordinary language. It underpins many business workflows, from recognising customer intent and routing support tickets to categorising complaints and assessing sentiment. This paper introduces The Classifier, a general-purpose natural language classifier with a suite of domain-specific adapters. The model scores a supplied set of candidate answers, allowing the general model to support different classification tasks while the adapters specialise it for particular domains and uses. The paper describes the data, training objective and architecture, followed by an internal evaluation on 4,264 held-out tasks. The general model achieved 81.3% top-1 accuracy, compared with 81.1% for Jev 1.13.0, although performance varied substantially across tasks. Intent classification was a particular strength, while Jev performed better on several smaller groups.

Contents

1 Classification in business workflows **1**

2 Data **2**

3 Objective **4**

4 Model architecture **4**

5 Training **6**

6 Results **8**

7 Inference architecture and performance **9**

8 Reflections **10**

1 Classification in business workflows

Natural language classification turns text into a decision over a set of possible categories. The input might be a customer message, a support ticket, a complaint or a review; the output

might identify an intent, a queue, a product, a cause or a sentiment. Although the vocabulary changes between workflows, the underlying problem is similar: interpret what the text means and determine which of the available choices best describes it.

A message asking to close an account, for example, needs to be recognised as a cancellation request before it can reach the appropriate workflow. A complaint may need to be classified by both the product involved and the reason for dissatisfaction. These decisions require an understanding of the text and the categories themselves, particularly where labels are ambiguous or their meaning depends on the business context.

The Classifier is a general-purpose natural language classifier designed for this problem, with a suite of domain-specific adapters built from a common general checkpoint. The general model provides a shared approach across classification tasks, while the adapters specialise it for intent, triage, sentiment, review stars, cause, codify and complaints. Both use the same input format and scoring method, so the classification question and available answers can change with the workflow.

Each decision is expressed as a record, a question and a menu of possible answers. Optional criteria explain what an answer means when a short label leaves room for interpretation. The Classifier ranks those choices in context, allowing the same model architecture to work with different label vocabularies rather than requiring one fixed set of categories across all datasets.

The evaluation examines this approach across banking, insurance and telecom intent, complaint products and causes, IT and HR ticket categories, sentiment, and review stars. It considers both the general model's performance across these tasks and the effects of specialisation through its adapters. Its scope is classification; generating replies or extracting open-ended answers would require a separate evaluation.

2 Data

The data comes from customer requests, support tickets, complaints, reviews and structured business records. The matrix below shows how the preparation code maps these sources to the seven Version 1.1 task domains. A tick means that a dataset is included in the preparation for that task, while a blank cell means that no mapping is defined. This describes the available task sets; establishing exactly what went into a particular checkpoint still requires its training manifest.

Dataset	Intent	Triage	Sentiment	Stars	Cause	Codify	Complaints
Bitext banking	✓						
Bitext insurance	✓						
Bitext telecom	✓						
CLINC150 (banking and out-of-scope)	✓						
CFPB complaints					✓	✓	✓
NYC 311						✓	
NYC HPD						✓	
IT support tickets		✓					
SAP HR requests[E]		✓					

Dataset	Intent	Triage	Sentiment	Stars	Cause	Codify	Complaints
Twitter US Airline Sentiment			✓		✓		
TweetEval sentiment			✓				
Yelp Review Full				✓			
Financial PhraseBank (all-agree)[E]			✓				

[E]Evaluation only: SAP HR and Financial PhraseBank contribute no training examples in this preparation. The general pooled task set includes each source once, even when it appears in several columns. The IT benchmark also links the Tasksource mirror.

Some sources support more than one kind of decision. A CFPB complaint can be classified by product, feeding the codify and complaints tasks, or by issue, feeding cause and complaints. The airline dataset similarly supplies both sentiment labels and reasons for negative experiences. Multiple ticks therefore reflect different questions that can be asked of the same records.

Earlier general training mix. The separate source manifest lists Banking77, Bitext customer support, Mozilla Smart Intent, Support Ticket Router, insurance claims processing, GoEmotions, Amazon Polarity, IBM Telco churn, customer-support CSAT. These earlier sources are not mapped to the seven specialist task sets above; their inclusion in a particular checkpoint requires its run manifest.

2.1 Reshaping records into choice menus

To bring these sources into a common format, each example is converted into a record, a question, a menu of candidate answers and the index of the correct answer. Dataset labels become readable choices, with optional criteria where their meaning needs explanation. The same record can then support several questions, such as which product a CFPB complaint concerns and what caused the complaint.

For example, the record “Please close my account immediately” can be paired with the question “What is the customer’s intent?” and the choices cancellation, billing, support and none/other. Cancellation is the correct answer. A sequence is constructed for each candidate using the same record, question and full menu, so the model learns to compare the answers in context. The complete menu, together with its correct-answer index, forms one training example.

The preparation retains the full supplied menu, including a fallback choice for out-of-scope records where the dataset supports one. Menu sizes consequently vary considerably: the audited intent training files contain 26, 30, 31 or 39 choices per example, whereas triage contains seven. Every candidate is scored with the complete menu as context, a detail that becomes important when comparing training times.

2.2 Splits and evaluation sampling

The split strategy follows the structure of each source. Bitext and airline sentiment use label-stratified 10% holdouts, while CFPB and NYC complaints use chronological holdouts. CLINC150, IT tickets, TweetEval and Yelp retain their publisher splits. Yelp training is capped at 20,000

reviews balanced across ratings, and SAP HR and Financial PhraseBank are reserved for evaluation.

From these held-out pools, the Version 1.1 benchmark samples up to 20 tasks per gold label using seed 7, producing 4,264 distinct tasks. These counts describe the benchmark rather than the larger training and preparation pools. Because some datasets contribute to more than one task group, the group counts overlap and cannot be added together. Dataset licences also vary by source, and complete training manifests remain to be recorded for each checkpoint.

3 Objective

Given a record x, a question q, and a menu $C = \{c_1, \ldots, c_K\}$, the training objective is to place the correct answer above the alternatives. Optional criteria help define the choices, and the model assigns each candidate a real-valued score z_i. These scores are converted into probabilities within the menu, with the loss determined by the probability assigned to the correct choice:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \qquad \mathcal{L} = -\log p_y,$$

where y indexes the correct choice. This menu-level cross-entropy trains the model to choose between the available answers, with padded choices masked before normalisation. Performance is measured primarily by top-1 accuracy: how often the highest-scoring answer is correct. Mean reciprocal rank provides a second view by taking account of where the correct answer appears when it is not ranked first.

Since the task supplies the answer vocabulary, the model can only select from the choices it is given. It can still choose incorrectly, and its probabilities must be read in the context of that menu: adding an answer or changing a criterion can alter the relative scores. Whether those probabilities are calibrated consistently across different tasks has not yet been established.

4 Model architecture

The model is built on Qwen3-1.7B-Base, a causal transformer with 1.7 billion parameters. Its model card describes 28 layers and grouped-query attention with 16 query heads and eight key/value heads. For classification, the backbone's hidden states provide a representation of each candidate input, which is passed to a scoring head.

Each input combines the record, question, full menu and candidate being scored. A learned linear head takes the final non-padding hidden state h_i and reduces it to a single score:

$$z_i = w^{\mathsf{T}} h_i + b.$$

This head is shared across all candidates and is stored and trained in float32, while the CUDA backbone is loaded in bfloat16.

4.1 How a choice becomes a score

During fine-tuning, the original language-model weights remain fixed while small LoRA adapters learn adjustments within selected layers. A scoring head learns alongside them, turning the resulting representation into a number that can be compared with the scores for other choices. Figure 1 follows one candidate through this process and distinguishes the frozen components from those that are trained.

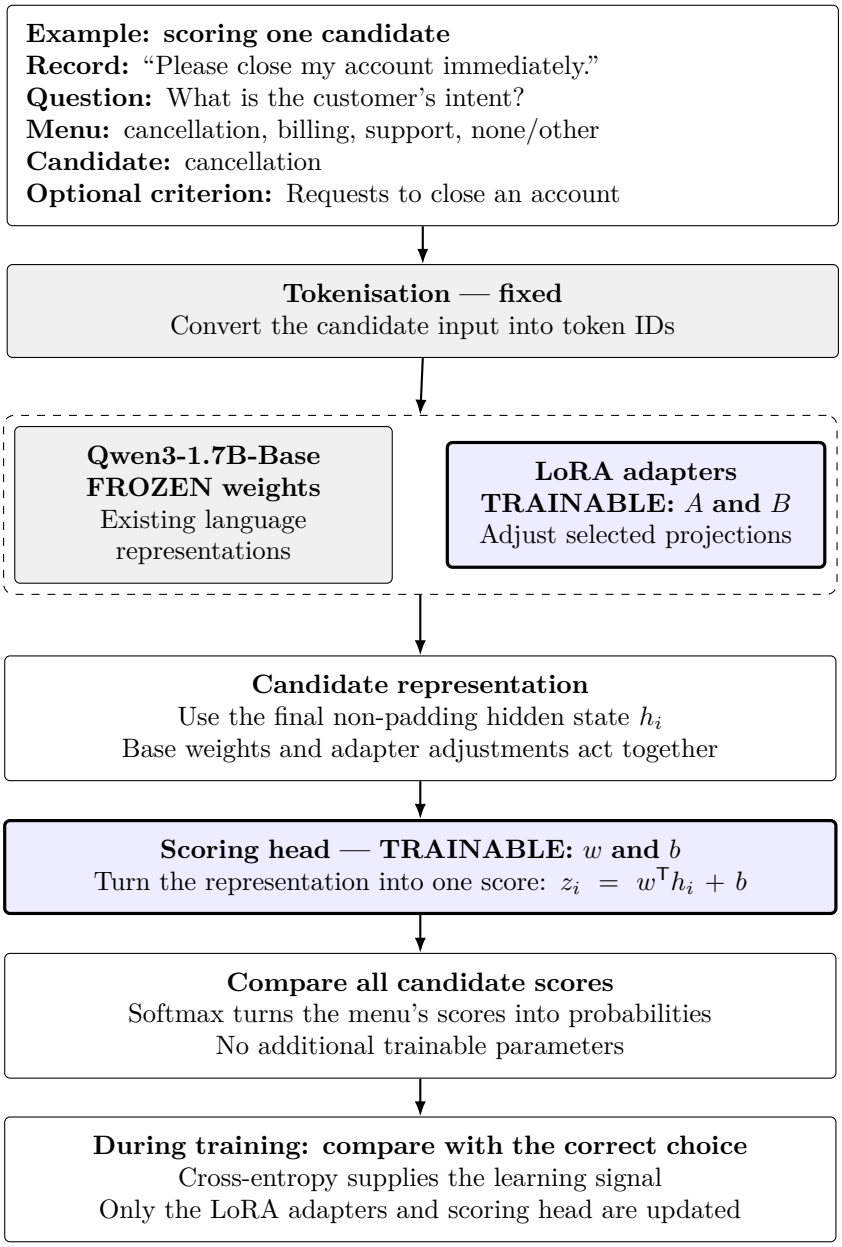


Figure 1: Candidate scoring and parameter updates. Shaded boxes labelled FROZEN retain their pretrained weights; heavy-bordered boxes labelled TRAINABLE are updated during fine-tuning. The two boxes inside the dashed outline are parts of the same backbone, not separate models. This scoring path is repeated for the candidates in a menu, sharing the same parameters. The final loss step is used only during training.

What "frozen" means. Freezing the base model leaves its role in processing the input intact; only updates to its stored weights are disabled. Learning is concentrated in the adapters and scoring head, which adjust how those existing representations are used for classification. Once training is complete, inference uses all of these weights in their fixed state and omits the comparison with the correct answer.

4.2 Low-rank adaptation

The seven specialist checkpoint configurations confirm LoRA through PEFT with rank $r = 16$, scaling parameter $\alpha = 16$, dropout 0.05 and no bias adaptation. In the standard LoRA formulation, a frozen projection W receives a trainable update:

$$W' = W + \frac{\alpha}{r} BA.$$

All seven specialist configurations apply these updates to `q_proj` and `v_proj`. Including the scoring head, the logs record 3,213,313 trainable parameters out of 1,723,788,289 in total, or approximately 0.186%. The specialists therefore differ in what they were trained on, while retaining the same number of trainable parameters.

These parameters are trained using the menu cross-entropy described in Section 3. Version 1.1 comprises the general model and seven task adapters—intent, triage, sentiment, stars, cause, codify and complaints—that specialise the same starting checkpoint. Their names indicate the intended use; the results below show why performance still needs to be checked against the task at hand.

5 Training

Training was carried out on a **single RTX 4000 Pro Blackwell GPU**, with each specialist initialised from the general Version 1.1 checkpoint and fine-tuned for one epoch. The configuration below combines values recorded in the workstation logs—batch size, gradient accumulation, sequence length, chunk size and precision—with the optimiser, learning-rate schedule and seed specified in the checked-in trainer. The latter settings were not explicitly recorded at run startup.

Setting	Specialist configuration
Optimiser	AdamW
Learning rate	2×10^{-4}
Schedule	Linear decay; 3% warm-up
Candidate-row batch target	4
Gradient accumulation	16 batches
Menus per full update in these runs	16
Epochs	1
Maximum sequence length	2,048 tokens
Random seed	7
Candidate chunk size	16
Checkpoint token threshold	2,048
CUDA backbone / head precision	BF16 / FP32

Although the sampler targets four candidate rows per batch, it keeps each menu together. In these task sets, that means one menu per batch: even two of the smallest, three-choice

sentiment menus would exceed the target. Gradients are accumulated over 16 batches before a full optimiser update, while candidate chunking and gradient checkpointing control memory use. The logs identify the attention implementation as `flash_attention_4`.

5.1 Training duration and compute accounting

The general Version 1.1 checkpoint took **23 hours 44 minutes** to train. The seven specialists were then fine-tuned from it one after another on 4 October 2026, using the same GPU. Their training adds a further 19 hours 36 minutes to the reported total, bringing the combined time to 43 hours 20 minutes.

Model	Training time
Version 1.1 (general)	23 h 44 min
Triage	3 min
Stars	31 min
Sentiment	1 h 28 min
Cause	54 min
Codify	1 h
Complaints	1 h 7 min
Intent	14 h 34 min
Specialists combined (reported)	**19 h 36 min**
General + specialists (reported totals)	**43 h 20 min**

The individual times above are rounded to the nearest minute. Summing the precise run intervals in the workstation logs gives 19 hours 36 minutes 27 seconds, which accounts for the apparent one-minute discrepancy in the rounded figures. These intervals include model loading and checkpoint saving between successive run markers, so they describe elapsed run time rather than GPU kernel execution alone. Peak memory and measured utilisation were not recorded.

Evaluation followed on 5 October 2026, after all seven specialists had completed training.

5.2 Why training times differ

The large spread in training times is mainly explained by the amount of candidate-scoring work. Each specialist has the same number of trainable parameters, but the number of records and the choices attached to each record vary substantially. The following table combines the startup counts with elapsed run intervals from `logs/train-from-v11.log` on the GPU workstation.

Adapter	Menus	Candidates	Choices/menu	Elapsed
Triage	1,572	11,004	7.0	2m 44s
Stars	20,000	100,000	5.0	30m 54s
Sentiment	60,567	181,701	3.0	1h 27m 30s
Cause	28,221	276,766	9.8	53m 53s
Codify	15,716	220,518	14.0	1h 0m 12s
Complaints	30,480	301,615	9.9	1h 7m 19s
Intent	84,598	2,668,124	31.5	14h 33m 55s

Compared with triage, intent contains 53.8 times as many menus and 242.5 times as many candidate sequences. Each menu also takes longer on average: approximately 0.620 seconds

compared with 0.104 seconds. Multiplying the difference in menu count by the difference in time per menu accounts for the roughly 320-fold gap in elapsed time. Wider menus explain much of the extra work, although token-length and execution profiling would be needed to isolate the remaining difference in cost per candidate.

The source composition helps explain those wider menus. Intent includes 22,685 Bitext banking menus, 23,161 telecom menus and 34,802 insurance menus, with 26 choices for banking and telecom and 39 for insurance. A further 3,600 CLINC banking menus and 350 out-of-scope menus have 30 or 31 choices. Triage, by comparison, consists of 1,572 IT-ticket menus with seven choices each; its HR data is used only for evaluation. The longer intent run thus reflects a much larger workload, without requiring a larger adapter or additional epochs.

6 Results

The benchmark contains 4,264 distinct held-out tasks, sampled with up to 20 tasks per gold label using seed 7. The Classifier 1.1 task adapters were evaluated on 5 October 2026 after one epoch of fine-tuning, following an evaluation of Jev 1.13.0 on the same tasks on 4 October. The available snapshot records the comparison scores but lacks enough detail about Jev's configuration to reproduce that baseline independently.

The results are reported as top-1 accuracy and mean reciprocal rank. The first measures how often the highest-scoring choice is correct; the second averages the reciprocal of the correct answer's rank, giving some credit when it appears near the top. Group scores are weighted by task count, and the overall total counts each dataset once even when it contributes to several groups.

6.1 Comparison and task specialisation

Overall, the general model achieved 81.3% top-1 accuracy and a mean reciprocal rank of 0.883, compared with Jev's 81.1% and 0.876. The task breakdown tells a more useful story than that narrow overall difference. Intent, which accounts for 59.1% of the evaluation, reached 97.5% accuracy against 94.2% for Jev, while triage and sentiment favoured Jev: 70.5% versus 80.8% for triage, and 71.1% versus 82.8% for sentiment.

Specialisation improved the intended task in some cases, but the gains did not necessarily carry across the evaluation. The intent adapter reached 98.2% on intent, above both the general model's 97.5% and Jev's 94.2%, while its overall accuracy fell to 79.4% from the general model's 81.3%.

The sentiment adapter showed a different pattern. Its sentiment accuracy rose modestly to 72.2%, from 71.1% for general, but remained below Jev's 82.8%. On review stars it reached 72.0%, ahead of both general at 68.0% and Jev at 71.0%. This is a useful reminder that the adapter's name alone does not identify where it will perform best.

For a business choosing a model, the most relevant comparison is therefore the task and dataset closest to the intended workflow. The group-level comparison below makes the general model's strengths and weaknesses visible alongside its overall score.

Group	Tasks	Jev 1.13.0	Classifier 1.1 general
All datasets	4,264	81.1%	**81.3%**
Intent	2,520	94.2%	**97.5%**
Cause	981	51.6%	**53.2%**
Complaints	894	**56.8%**	51.8%
Codify	249	**68.3%**	51.0%
Triage	234	**80.8%**	70.5%
Sentiment	180	**82.8%**	71.1%
Stars	100	**71.0%**	68.0%

Groups overlap; task counts must not be summed. Bold marks the higher of the two reported scores, not statistical significance.

7 Inference architecture and performance

At inference time, the trained backbone and scoring head assign a score to each candidate, and those scores are normalised over the menu. Because the answer is selected from the supplied choices without autoregressive generation, the work depends on the input length and the number of candidates to score. The inference implementation developed around reducing repeated work and sharing the backbone across specialists.

7.1 Stage 1: direct candidate scoring

The simplest implementation constructs a complete input for each candidate, runs it through the model, and compares the resulting scores. In the example in Figure 1, that means processing the account-closure message, question and menu once for cancellation, again for billing, and then for support and none/other. This provides a straightforward reference implementation, but much of the text is identical on every pass.

7.2 Stage 2: shared-prefix caching

Shared-prefix caching addresses that repetition. The record, question and full menu form a common prefix, followed by a suffix identifying the candidate answer. By retaining the prefix's attention keys and values, the scorer can reuse that computation as it processes each candidate suffix. The opportunity to save work is greatest when the shared record is long relative to the individual answers.

The amount of reuse also depends on how candidates are batched. Chunking limits memory use, but the checked-in routine can recompute the prefix for each chunk, so it does not always process the shared text only once per menu. Input length, menu size, suffix length and chunk size all affect the resulting gain. The cache also belongs to the model state that produced it, which means a change of adapter requires compatible prefix computation to be performed again.

7.3 Stage 3: loading and switching specialist adapters

Supporting several specialists introduces another opportunity to share work. Instead of loading a complete backbone for every model variant, the serving implementation keeps one base model and loads the approved LoRA adapters together with their scoring heads. Each scoring operation

activates the selected adapter and its matching head, and a newly approved checkpoint can be loaded into the process while it is running.

Loading a new adapter still requires its weights and head to be read and placed in memory, whereas switching to a resident adapter avoids that loading step. Hot-loading therefore saves the cost of reloading the full backbone without making switching free or allowing an unlimited number of resident adapters. Because the active model state is shared, adapter selection and scoring must remain coordinated; different adapters cannot simply be run in parallel against the same mutable state.

Together, these changes reduce different kinds of repeated work: caching shares the input computation across candidates, while adapter loading shares the backbone across specialists. The available record describes this progression but does not contain a controlled measurement of the speed-up at each stage.

7.4 Performance evidence

The project's capacity configuration gives reference GPU service times of approximately 40 ms for a 2,150-input-token example and 109 ms for an 11,750-input-token example. These provide context for the intended workload, but the record does not establish the menu sizes, timing method, inference hardware or warm-up conditions behind them. They cannot therefore be treated as a reproduced benchmark on the training GPU, or as end-to-end latency or p50/p95 measurements.

A controlled evaluation would compare shared-prefix caching with full candidate scoring on identical inputs, using fixed hardware and numerical precision. Reporting menu sizes, token lengths, latency distributions, throughput and peak memory would make the results interpretable, while separating model execution from queueing and transport would show where time is actually spent. That comparison remains to be carried out.

8 Reflections

The Classifier brings a general-purpose model and a suite of domain-specific adapters to the problem of natural language classification. Its shared scoring approach accommodates the labels and criteria of different business workflows, while the supplied menu keeps the answer within categories the business can act on. The quality of the resulting decision depends both on the model and on how clearly the question and choices are defined.

The evaluation suggests particular strength in intent classification, alongside clear weaknesses in other tasks. The overall difference—81.3% for general against 81.1% for Jev—is too narrow to establish a meaningful general advantage without confidence intervals or repeated runs, especially when intent accounts for 59.1% of the tasks. A useful deployment evaluation should therefore reflect the business's own data and failure modes.

The specialist results reinforce that point. Fine-tuning the intent adapter improved intent accuracy while reducing its overall score, and the sentiment adapter performed best among the listed models on review stars. Selecting an adapter by measured performance on the intended task is consequently more reliable than choosing the closest-sounding name.

Further work should make the experiments easier to reproduce and test how well these results carry into independent business data. Complete run manifests, precise timing and memory records, repeated evaluations and calibration would strengthen the evidence. Small groups such as the six NYC HPD tasks need particular care, while training-source overlap, label wording,

sampled menus and domain shift remain possible influences on performance. Where real records may fall outside the supplied categories, the fallback choice also needs to be evaluated explicitly.

Reproducibility sources. The architecture and optimisation details above come from `train/train_lora.py`; data preparation from `train/prepare_domains.py` and `data/APACHE_SOURCES.txt`; numerical results from `web/app/models/benchmark.js`. The online evaluation tables retain per-dataset results for all eight adapters. This revision documents existing experiments; it does not introduce a new training run or performance measurement.