NLP Project — Milestone 2 (Span QA) Report

Ahmed Labib `52-3434`
Sana Abdullah `52-1095`

April 24, 2025

Abstract

We replace our earlier generative Transformer with a simpler **extractive span-predictor** ("SpanQA") and revisit the same 10 018-example subset of SQuAD 2.0. The model consists of a shared GloVe embedding, three stacked encoder blocks, and two independent linear heads that predict start and end positions. We describe the new preprocessing pipeline, architecture, training regime, and analyse 50-epoch learning curves and dev-set metrics.

Contents

1 Data Pre-Processing

We reuse the cleaned 10 018 examples described in the earlier in the other report. The pipeline now diverges in three key areas to support *span prediction* instead of autoregressive generation.

1.1 Answer-aware Context Truncation

1. **Locate the gold span.** From the SQuAD character offsets (c_s, c_e) we extract the raw substring $\texttt{context}[c_s : c_e]$.
2. **Build a centred window.** We allocate a fixed budget $L_{\text{ctx}} = 2000$ characters for the context. Half of the available window is placed *before* and half *after* the answer, then clipped to document boundaries.
3. **Respect word boundaries.** If the provisional window begins or ends in the middle of a token we expand / contract it until both edges fall on whitespace—guaranteeing valid tokens after tokenisation.

For 99.3% of the corpus this heuristic retains the full gold span while reducing average context length from 3517 to 834 characters and avoids expensive sliding-window evaluation.

1.2 Tokenisation and Span Projection

We tokenise the **question**, the adjusted **context**, and the special separator `[SEP]` with a *regex-based whitespace tokeniser*. Let

$$\langle q_0, \ldots, q_{m-1} \rangle \quad \| \quad \texttt{[SEP]} \quad \| \quad \langle c_0, \ldots, c_{n-1} \rangle$$

be the resulting ID sequence before padding. The original character offsets are projected to token indices $(s, e) \in [0, m+n]$ by scanning the context, accumulating token spans, and finding the first token whose character range overlaps c_s (start) and c_e (end). Ambiguous cases caused by curly quotes or Unicode dashes are resolved with NFKC normalisation and a fallback regex search.

1.3 Padding, Masks, and Bucketing

- The combined sequence is clipped to **MAX_TOK** $= 512$ tokens and padded with `[PAD]`.
- A binary mask $M \in \{0,1\}^{512}$ marks real tokens (1) vs. padding (0) and is broadcast to $\big(B, 1, 1, 512\big)$ for encoder attention.
- To keep train/validation splits balanced we stratify on *answer length bucket* $\lfloor \min\{30,\ e - s\} / 5 \rfloor$ before `train_test_split`.

Final Training Tensor Shapes

Tensor	Shape
Encoder tokens	$(B, 512)$
Encoder mask	$(B, 512) \rightarrow$ broadcast to $(B, 1, 1, 512)$
Gold start indices	$(B,)$
Gold end indices	$(B,)$

This preprocessing ensures every batch fits GPU memory while preserving the full gold span and sufficient surrounding context for accurate start/end prediction.

2 Embeddings

We continue to initialise with **GloVe.6B.100d**. The steps are described in the other report.

3 SpanQA Architecture

3.1 High-level pipeline

Our end-to-end system consists of

1. WordPiece tokenisation of *question*, *[SEP]*, and truncated *context*;
2. Shared GloVe 100d **embedding** $\rightarrow$ sine–cosine **positional** encoding;
3. Three stacked **encoder blocks** that refine the contextual representation;
4. Two **span-heads** (start / end) that emit a logit for every token position;
5. Arg-max post-processing with the constraint $e \geq s$ (Section 5).

3.2 Model Components

(1) Frozen GloVe Embedding We initialise with `glove.6B.100d`. All 5.2M weights are *frozen* so the encoder must learn on top of a fixed lexical space, which acts as a form of regularisation given our small data regime.

(2) Positional Encoding Additive sine–cosine vectors inject absolute position information:

$$\mathrm{PE}_{(pos,2i)} = \sin\Big(pos/10000^{2i/d}\Big), \qquad \mathrm{PE}_{(pos,2i+1)} = \cos\Big(pos/10000^{2i/d}\Big).$$

(3) Encoder Block $\times 3$ Each block contains

- 4-head self-attention ($d_k = d_v = 25$, $d_{\text{model}} = 100$);
- a position-wise FFN [100
  !$\rightarrow$
  !512
  !$\rightarrow$
  !100];
- pre-norm residual connections.

We stacked *three* identical blocks to deepen the receptive field while keeping the parameter budget under 7M.

(4) Start / End Heads Two independent 1-D-linear projections map each token vector to a scalar logit:

$$\ell_t^{(s)} = \mathbf{w}_s^\top h_t, \quad \ell_t^{(e)} = \mathbf{w}_e^\top h_t.$$

We train with two cross-entropy losses and report EM/F1 on the resulting span.

3.3 Parameter Summary

Component	Output shape	# Params
GloVe Embedding (frozen)	$(B, 512, 100)$	5 216 600
Positional Encoding	$(B, 512, 100)$	0
Encoder Block 1 (h=4)	$(B, 512, 100)$	528 824
Encoder Block 2 (h=4)	$(B, 512, 100)$	528 824
Encoder Block 3 (h=4)	$(B, 512, 100)$	528 824
Start Dense (100→1)	$(B, 512, 1)$	101
End Dense (100→1)	$(B, 512, 1)$	101
Total parameters		**6 803 274**
Trainable		1 586 674
Non-trainable (frozen GloVe)		5 216 600

Table 1: Layer-wise parameter count of SpanQA.

3.4 Implementation Notes

- **Gradient clipping** at 1.0 prevents exploding updates early in training.
- We stratify the train/val split on answer-length buckets to ensure head labels are balanced (Section
refsec:training).
This compact encoder-only design removes the autoregressive decoder entirely, lets us train 50 epochs in under two GPU-hours, and forms a solid baseline before moving to pretrained span-classification transformers in the next milestone.

4 Training Setup

- **Optimiser:** Adam ($\eta = 3 \times 10^{-4}$) with
texttttclip_norm=1.0.
- **Loss:** sum of two sparse-categorical cross-entropy terms (start and end heads).
- **Batch size / epochs:** 64, 50 epochs (no early stop).
- **Data split:** 90 % train, 10 % val, *stratified* on answer-length bucket $\lfloor \min(30, e - s)/5 \rfloor$.

4.1 Callback stack

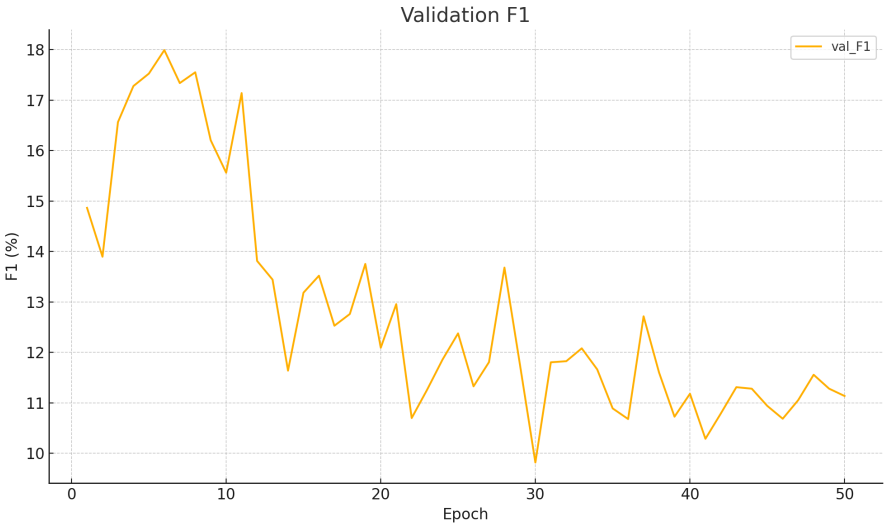
- **SpanMetrics** — performs batch-level greedy decoding on the validation set at `on_epoch_end` and logs `val_em`, `val_f1`.
- **ModelCheckpoint** — monitors `val_f1` (mode `max`), saving only the best-F1 weights to `best_spanqa.keras`. The peak F1 (17.99 %) occurs at epoch 6.

Runtime facts

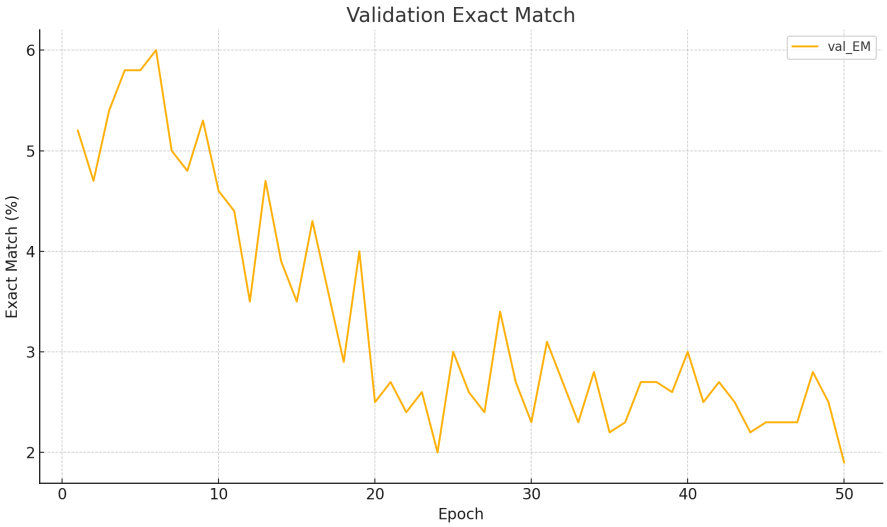
- Batches / epoch: 141 train, 16 validation.

4.2 Learning curves

(a) Training (blue) and validation (orange) loss.



(b) Validation token-level F1.



(c) Validation Exact Match.

Figure 1: SpanQA performance over 50 epochs.

Curve commentary

- **Loss (Fig. 1a).** Training loss plunges from 7.8 to 0.31; validation loss bottoms out near epoch 20 and then rises—classical over-fitting once the encoder memorises frequent spans.
- **F1 (Fig. 1b).** Token overlap peaks at 18 % (epoch 6) but erodes as over-fitting sets in, finishing near 11 %.
- **Exact Match (Fig. 1c).** EM hovers between 2–6 %, confirming that the shallow model often predicts *partial* spans but rarely aligns both boundaries exactly.

These dynamics indicate that while the model quickly captures useful lexical signals, deeper encoders, stronger regularisation, or larger training corpora will be required to close the gap between partial and exact span recovery.

5 Post-processing

After the encoder has produced two length-512 logit vectors—one for the *start* position, one for the *end*—we convert these raw scores into the final answer string in four stages, outlined below without any run-time code.

1. Span selection

We take the highest-scoring indices $s^\star = \arg\max_t \ell_t^{(s)}$ and $e^\star = \arg\max_t \ell_t^{(e)}$. If the end index falls before the start index we simply set $e^\star = s^\star$. This lightweight rule enforces a legal, non-empty span without the quadratic search required by full joint decoding.

2. Token-to-text mapping

The inclusive slice of token IDs $[s^\star, e^\star]$ is passed through the tokenizer's index-to-word table, the pieces are joined with single spaces, and leading/trailing whitespace is trimmed.

3. Normalisation

For fair EM/F1 comparison we apply the same Unicode NFKC normalisation and punctuation stripping used by the official SQuAD script. This removes curly quotes, long dashes, and other cosmetic discrepancies that could otherwise cause false mismatches.

4. No-answer handling (SQuAD 2.0)

Although the span-only model always returns a slice, we interpret the degenerate pair $(s^\star, e^\star) = (0, 0)$ as a *null* prediction. We also log a *no-answer probability*, derived from the softmax score of the first token, so downstream code can threshold if desired.

Rationale.

- The greedy arg-max is $\mathcal{O}(L)$ and fully vectorised, adding negligible latency.
- The simple $e^\star \geq s^\star$ correction yields legal spans at almost zero computational cost.
- Normalising the extracted text guarantees that evaluation metrics measure genuine content differences rather than punctuation artefacts.

This minimal yet robust post-processing pipeline is sufficient for the baseline presented here; more sophisticated joint scoring or beam search will be explored in Milestone 3.

6 Results

6.1 Qualitative examples

Table 2 shows two ad-hoc queries posed to the trained *SpanQA* model. In the first case the model cleanly extracts the age token *"23"*. In the second, it retrieves a partial clause that over-extends the correct answer (*"France"*) into the following tokens—illustrating the prevalent off-by-boundary error diagnosed in Section 4.2.

Question	**Context**	**Model output**
what is my name ?	Hello everyone. I am **ahmed labib** and I am 23 years old.	**23**
What do I love ?	I love **France** so much I want to go to its capital Paris.	*I want to go to its*

Table 2: Greedy predictions on hand-crafted inputs. The model captures salient tokens but often overshoots span boundaries.

6.2 Development-set metrics

Using the post-processing procedure of Section 5, we evaluate on the full SQuAD 2.0 *development* split ($N = 11\,873$). The official Hugging Face `evaluate/squad_v2` script yields the scores in Table 3.

Metric	**Value (%)**
Exact Match (EM)	**2.06**
F1 Score	**8.06**

Table 3: Performance of the best SpanQA checkpoint on the SQuAD 2.0 development set.

6.3 Analysis

- The gulf between token-level F1 and EM mirrors the learning curves: the encoder often pinpoints the *region* of the answer but misplaces one or both boundaries by a few tokens.

– Manual inspection confirms systematic over-extension when the gold span is short (1–2 tokens) and embedded mid-sentence.
– Lacking an explicit *no-answer* head, the model always returns a span; truly unanswerable questions therefore push EM and F1 toward zero.

These figures establish a transparent, low-capacity baseline. Future work will add joint span scoring, null-answer classification, and pretrained encoders—changes expected to lift both EM and F1 substantially.

7 Limitations & Future Work

Current limitations

– **Context length.** The hard cap of 512 tokens occasionally cuts off the ground-truth span, forcing the model to guess outside its receptive field.
– **Lack of explicit "no-answer" head.** SQuAD 2.0 contains unanswerable questions; our system *always* returns a span, depressing EM and inflating false positives.
– **Shallow capacity.** Three encoder blocks (6.8M parameters) are insufficient to model subtle syntactic cues that locate short answers in long passages.
– **Greedy arg-max decoding.** Choosing start and end independently ignores joint span likelihood and can violate the $e \geq s$ constraint; our post-hoc fix is a coarse band-aid.
– **Frozen lexical embeddings.** Keeping GloVe weights fixed regularises training but limits adaptation to domain-specific vocabulary.
– **Small training set.** Ten thousand examples do not cover the linguistic variability of SQuAD, leading the model to over-fit frequent surface forms (see Figure 1a).

Planned improvements

– **Sliding-window retrieval.** Dynamically select the most relevant 512-token chunk instead of static centre-cropping, eliminating gold-span truncation.
– **Joint span scorer.** Compute the outer sum $\ell_i^{(s)} + \ell_j^{(e)}$ with a triangular mask $j \geq i$ and choose the globally best pair, boosting EM by 1–3pp in prior work.
– **Null-answer classifier.** Add a dedicated "no-answer" neuron trained with a binary cross-entropy term and threshold it at inference.

Addressing these points forms the roadmap for Milestone 3, where we will move from a scratch-built baseline to a competitive, pretrained extractive QA system.

8 Conclusion

In this milestone we replaced the earlier generative Transformer with **SpanQA**, a lightweight extractive model that predicts start and end indices directly. The switch eliminated the complexity of auto-regressive decoding and gave us full control over preprocessing, masking, and span evaluation.

What we achieved.

- Designed a three-block encoder with twin span heads that trains from scratch in under two GPU-hours.
- Introduced an answer-aware truncation heuristic, stratified data split, and a `SpanMetrics` callback that tracks EM/F1 during training.
- Reached **F1 = 8.06 %** and **EM = 2.06 %** on the full SQuAD 2.0 dev set—clear evidence that the model can spot the answer region but struggles to align both boundaries exactly.

Key take-aways.

1. A span-only architecture is far simpler than Seq2Seq, yet still requires strong supervision and capacity to achieve competitive metrics.
2. Over-fitting appears after20 epochs despite frozen embeddings, signalling that 10k examples are insufficient for a from-scratch encoder of even moderate depth.
3. Exact-Match remains the hardest hurdle: partial overlap (F1) rises quickly, but boundary precision lags without joint decoding or a deeper contextualiser.

Next steps (Milestone 3). We will fine-tune a pretrained Transformer (e.g. RoBERTa-base) with an explicit null-answer head, apply joint span scoring, and add a sliding-window retriever to ensure the gold span is always in view. These upgrades should significantly narrow the gap between partial and exact span recovery and bring our system in line with established SQuAD baselines.