National College of Ireland

SPELLS Configuration Manual

MSc Research Project
MSc in Artificial Intelligence

Sam Conneely
Student ID: X24157651

School of Computing
National College of Ireland

Supervisor: Dr. Muslim Jameel Syed

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Sam Conneely
Student ID:	X24157651
Programme:	MSc in Artificial Intelligence
Year:	2025
Module:	MSc Research Project
Supervisor:	Dr. Muslim Jameel Syed
Submission Due Date:	11/12/2025
Project Title:	SPELLS Configuration Manual
Word Count:	2015
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	11th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

SPELLS Configuration Manual

Sam Conneely
X24157651

1 Introduction

SPELLS: Semi-Supervised Power Estimation and Label Learning System for Trading Card Games - is a system responsible for the learning of the latent power of Magic: The Gathering cards using the features encoded directly on the cards themselves. SPELLS makes use of multimodal learning to return power levels for each of the cards within Magic: The Gathering.

This configuration manual goes through the data pipeline for SPELLS, explaining each of the components and how to run the code responsible for getting the results seen in the accompanying report.

2 Directory Structure and Required Files

This section gives a brief breakdown of what the directory structure and the files included should look like. Outlining the files required in order to run the code successfully. To run SPELLS, you must begin pulling the card data from the Scryfall API. Once this is done the images required need downloading before the cleaning and preprocessing pipeline can start. The files required to download all of the data needed are shown below.

2.1 Required Files

- mtg_cards_pipeline.ipynb
- download_images.py

The card data and image files downloaded from the Scryfall API are quite large but only need to be downloaded once initially. The card data is extracted from the json file downloaded and stored in a parquet file called cards_clean.parquet, which can be loaded later for quicker access to the data. All of the images downloaded using the multi-threaded python script are stored in the directory called card_images_cropped and are only used once when generating image embeddings.

2.2 Directory Layout Required

```
SPELLS/
 mtg_cards_pipeline.ipynb
 download_images.py

 card_images_cropped/
```

3 Hardware and System Requirements

When downloading the images the download_image.py script makes use of multi-threading to accelerated the process. Currently the max_workers variable in the file is set to 8. This sets the number CPUs to utilise for downloading the files.

The SPELLS pipeline includes multiple steps where SBERT is utilised to generated feature embeddings for downstream processing and training. The generation of these embeddings often take some time to run so the below requirements are suggested. The use of PCA, K-Means Clustering and Label Propagation also add to the number of time taken to run the SPELLS pipeline from start to finish.

3.1 Minimum Requirements

- 16GB Ram
- > 10GB disk space for datasets and image files
- Python 3.10+

3.2 Recommended Requirements

- Nvidia GPU with 6-12GB VRAM for acceleration
- CUDA compatible environment
- Fast NVMe storage

4 Software Requirements

SPELLS makes use of a number of python libraries to run the project. A number of libraries are used for downloading the images needed for embedding generation, also for handling any error when downloading the initial datasets required. The following list of libraries is required to reproduce the SPELLS configuration environment and can be found in the accompanying requirements.txt file:

Core Dependencies

- numpy 1.26.4
- pandas 2.2.2
- scipy 1.13.1

Machine Learning

- scikit-learn 1.5.1
- xgboost 2.1.0
- tensorflow 2.15.0
- keras 2.15.0

- umap-learn 0.5.5

Embedding Models

- sentence-transformers 2.7.0
- transformers 4.41.2
- torch 2.2.2
- tokenizers 0.19.1

Image Processing

- pillow 10.3.0
- opencv-python 4.9.0.80

Data Downloading Utilities - Used by the download_image.py file

- requests 2.31.0
- tqdm 4.66.4 - Progress bars

File Formats

- pyarrow 16.1.0

Visualisations

- matplotlib 3.9.0
- seaborn 0.13.2

Notebook Support

- ipykernel 6.29.4

Installation Commands

To run the requirements.txt file for installing all the the libraries needed to get SPELLS working correctly, you will need to enter the follow command into the terminal.

```
pip install -r requirements.txt
```

5 Configuration of the SPELLS Pipeline

This section of the manual explains each stage throughout the pipeline of the project and how to run it. There are multiple parts of the SPELLS pipeline that are required and they must be run in the correct order to produce the same results as are seen in the accompanying report. The steps listed below should be followed in sequence to replicate the entire process from start to finish.

5.1 Data Acquisition

The first step in the process is the downloading of the raw card data through the Scryfall API. Using the Scryfall URL seen in listing 1, the requests library reaches out to the API and pulls the required data

Listing 1: Scryfall bulk download URL

```
bulk_url = https://api.scryfall.com/bulk-data
```

The JSON file downloaded after running the function seen in listing 2 contains a dump of all card data available from the Scryfall website. This file a bulk download with all cards from all formats and sets in all languages, this data requires a lot of clean up before processing can begin.

Listing 2: Download Scryfall bulk JSON

```
def download_scryfall_bulk_json(save_path='default_cards.json'):
```

Once this JSON file has been downloaded and some basic cleaning up has been done, like only the English data is kept and all duplicates are removed. The now somewhat processed data is saved in parquet and csv formats for later use, and to avoid re-downloading the data each time. The storage of the processed data can be seen in listing 3.

Listing 3: Storing processed JSON data to parquet and csv format

```
df.to_parquet('cards_clean.parquet', index=False)
df.to_csv('cards_clean.csv', index=False)
print("Saved to 'cards_clean.parquet' and 'cards_clean.csv'")
```

5.2 Oracle Text Cleaning and Feature Engineering

Each of the modalities used throughout SPELLS required detailed preprocessing and normalisation before any embedding vectors could be generated for downstream training. The cleaning of the oracle text data is a three step process and is documented in detail throughout section 3.2.3 of the accompanying report.

The main areas of configuration within this process are the addition or removal of any colours within the MANA_MAP constant or keywords that could potentially be found on cards, seen within the EVERGREEN_KEYWORDS constant. Each of the two can be seen in listing 4 and 5. All explanation text found in the oracle text for each card is also removed using the `remove_extra_text()` function. Throughout this process abilities found of the cards are also tagged and saved as a new column within the master DataFrame being utilised for preprocessing.

Listing 4: Mapping of the colours and words can are found in the oracle text data

```
    MANA_MAP = {
    'W': 'white', 'U': 'blue',
    'B': 'black', 'R': 'red',
    'G': 'green', 'C': 'colorless',
    'X': 'x', 'T': 'tap', 'Q': 'untap'
}
```

Listing 5: List of evergreen keywords that are found on cards

```
EVERGREEN_KEYWORDS = ['cacade', 'bolster', 'flash', 'attach',
    'counter', 'exile', 'fight', 'equip'
"flying", "first strike",'protection', "double strike", "
   lifelink", "trample", 'multikicker',
"deathtouch", "vigilance", "menace", "haste", "hexproof", '
   delve', 'defender', 'enchant'
"indestructible", "reach", "prowess", "ward", 'mill', '
   sacrafice', 'scry', 'annihilator',
'bestow', 'crew', 'convoke', 'cycling', 'flashback', 'kicker'
   , 'manifest', 'mentor', 'morph',
'proliferate', 'renown', 'riot', 'surveil', 'umbra armor', '
   landfall', 'shroud']
```

5.3 Image Downloading

To download the images needed to generate the visual embeddings the download_images.py script makes use of the image_url column for each row of data within the master DataFrame stored as a parquet file. The url contains the version information for each of the cards storage within the dataset. This example of the image_url for the art_crop version of the card art `https://cards.scryfall.io/art_crop/front/0/0/0000419b-0bba-4488-8f7a-6194544ce91e.jpg?1721427487`. There are multiple different version of card art available from Scryfall but the cropped version was used as it did not contain any extra card information that might have been leaked into the embedding space for other features, for example some of the oracle text may have been learned by the CNN if the full card art was used.

All images pulled from Scyrfall are stored within the card_image_cropped folder as mentioned in section 2.2 - Directory Layout Required 2. The call to the download_image() function can been seen here `download_images(df, output_dir="card_images_cropped", log_file="download_errors.csv")`. This function starts off the downloading of all of the cropped card art images for any url found within the card_clean.parquet file that stores all of the card data pulled and cleaned previously `df = pd.read_parquet("cards_clean.parquet")`.

The expected number of images downloaded for the cropped art card is currently 26,795. The images are between 50kb and 150kb in size and the total file size is 2.15GB. See two examples of the cropped card art in figure 1. Each of the files for a card's url will be downloaded and saved with the within the card_images_cropped directory, with the card's id and name as the file name.

5.4 Embedding Generation

A number of different embeddings are generated for different modalities and features within the dataset. The oracle text data is the first modality for which the embeddings are generated. Once cleaned and normalised, the pretrained SBERT model is used to generated 384-dimension embeddings. The second modality to have embeddings generated is the images for all 26,795 cards within the set. A ResNet50 CNN is used to create the embeddings for each image. Even when using multi-threading, for all 419 batches to

Figure 1: Example of the cropped card art without borders. Cropped images contain no extra information that might cause data leaks downstream.

process it took between 15 and 30 minutes to generate the embeddings with the output expected to be an array of shape (26795, 2048).

```
# Run threaded and batched embedding generation
embeddings_dict, image_embeddings = get_image_embeddings_streamed
(cnn_model, image_folder, batch_size=64, max_workers=10)
```

5.5 Label Propagation

To run the label propagation step in SPELLS, a set of manually labelled cards was needed. These cards were selected using a stratified method, creating buckets of cards with a diverse spread of cmc values, card colours, rarity and whether or not the card was a creature. Two strata were sampled from within each of the buckets giving a total of 605 groups and 1113 cards within the final sample. These cards were saved to a constant called OUTPUT_CSV and stored in the csv file seen below.

```
INPUT_CSV = "mtg_labeling_data.csv" # Clean dataset
OUTPUT_CSV = "cards_to_label.csv" # Cards selected for manual
   labeling
SAMPLES_PER_GROUP = 2 # Number of cards per strata group
MIN_CMC = 0
MAX_CMC = 30
```

This data was then loaded into the next step in the label propagation process. These a subsample of these cards was then selected using K-Means clustering to get the final 50 cards that were to be manually labelled. The configuration setup for this can be seen below, with the number of clusters (cards) being mutable.

```
STRATIFIED_SAMPLE_CSV = "cards_to_label.csv"
FUSED_EMBEDDINGS_NPY = 'fused_embeddings_stratified.npy'
OUTPUT_CSV = "cards_to_label_final.csv"
NUM_CLUSTERS = 50
RANDOM_STATE = 42
```

A piece of helper code was then written to allow for the reading of the saved csv files and manual rating of each of the cards selected by K-Means clustering to be given a power score through the terminal. This helped to speed up the process and reduced the possibility of human error if done by hand. The name, id and oracle text was displayed to the terminal and then a score was inputted and stored as the cards power_score going forward. These scores were taken as the manual labelled for future processing.

```
----------------------------------------
Card 15/50
Name : Savage Gorilla
Oracle text:
{U}{B}, {T}, Sacrifice this creature: Target creature gets
             -3/-3 until end of turn. Draw a card.
----------------------------------------
```

Once this manual labelling process was complete, the label propagation stage could being. A constant, M was set up to set the number of anchor points within the dataset. The final number of anchors used in SPELLS was $M = 800$. K-Means clustering was then once again used to get the number of neighbours per anchor point.

After the label propagation process was complete, a brief check of the new column, propagated_power is done to see if the numbers generated through propagation look correct overall. The describe() function checks some basic details each of the cards new values.

```
count    26795.000000
mean         5.551374
std          1.621571
min          1.000000
25%          4.447919
50%          5.539628
75%          6.671761
max         10.000000
Name: propagated_power, dtype: float64
```

5.6 Active Learning Loop

As there as some outliers found to have strange values after the propagation, an iterative process of human correction was done to manually fix some of these values. The correction of the outliers had a major effect on the performance of the overall propagated results as it stopped many cards getting their values pulled off towards local anchors with incorrect scores. The helper function called `def card_score()` was created and used for manual corrections during the active learning process. The grading rubric was once again followed here when labelling cards manually, each step in the rubric had a weighting that gave each card it's final score.

```
weights = {
"mana_eff": 0.20, "impact": 0.20, "long_term": 0.20,
"flexibility": 0.15, "synergy": 0.15, "restrictions": 0.10}
```

5.7 Model Training Configuration

After the label propagation process is complete the running of the training process is simple and each of the cells can be run in sequence to complete the training of the three machine learning models. The performance of each of the models are outputted after the model is trained, this is the first pillar of evaluation as mentioned in the accompanying report.

5.8 Evaluation Configuration

The final step in the methodology of SPELLS in the evaluation process. This stage (pillar 2 of evaluation) makes use of K-Mean clustering to pick out 30 cards that represent the centroid values of each cluster, based on certain card archetypes 3. The final number of manually labelled cards within the SPELLS pipeline is 106 and based on these labels, 30 clusters are created and can be seen for both the fused embeddings with and without the image data 2. The number of clusters created (cards selected) can be modified by updating the constant `N_CLUSTERS` within the *Evaluation of the Three Models Trained* section in the SPELLS pipeline in mtg_cards_pipeline.ipynb file.

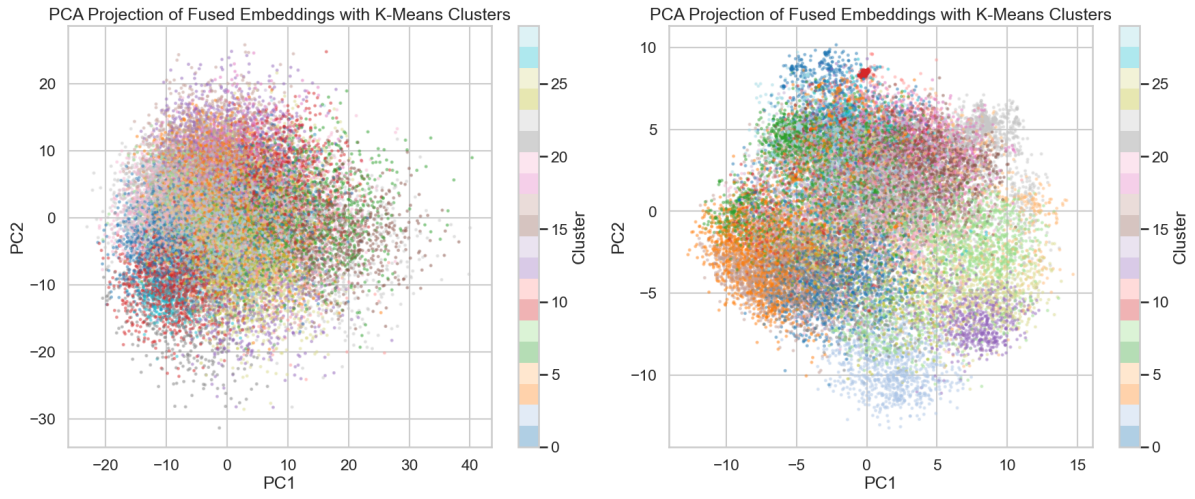

Figure 2: Plots showing the clusters for fused embeddings created as representative cards for the second pillar of evaluation. Both the embeddings here are with and without the image embedding data as tested in the paper. The plot shown on the left is the visualisation containing the image embeddings data. The plot of the right has had these embeddings removed for ablation testing.

6 Appendix

```
# Core Python utilities
numpy==1.26.4
pandas==2.2.2
scipy==1.13.1

# Machine Learning
```

Figure 3: Both fused representation spaces with centroid values of cards chosen for the second pillar of evaluation within the SPELLS pipeline.

```
scikit-learn==1.5.1
xgboost==2.1.0
tensorflow==2.15.0
keras==2.15.0
umap-learn==0.5.5

# Embeddings / NLP
sentence-transformers==2.7.0
transformers==4.41.2
torch==2.2.2
tokenizers==0.19.1

# Image Processing
pillow==10.3.0
opencv-python==4.9.0.80

# Downloading and Utilities
requests==2.31.0
tqdm==4.66.4

# Data Formats
pyarrow==16.1.0

# Visualization
matplotlib==3.9.0
seaborn==0.13.2

# Jupyter Notebook Support
ipykernel==6.29.4
```