National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

NagaRavali Ujjineni
Student ID: 24138312

School of Computing
National College of Ireland

Supervisor: Dr Giovani Estrada

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	NagaRavali Ujjineni
Student ID:	24138312
Programme:	Programme Name
Year:	2025-26
Module:	MSc Research Project
Supervisor:	Dr Giovani Estrada
Submission Due Date:	28/01/2026
Project Title:	Configuration Manual
Word Count:	642
Page Count:	7

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:	NagaRavali Ujjineni
Date:	28th January 2026

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):	

Configuration Manual

NagaRavali Ujjineni
24138312

1 Introduction

The given configuration manual has step-by-step instructions to recreate the study of Enhanced Kubernetes YAML Generation using Dual-Encoder CodeT5 with Actionable Feedback. The system transforms natural language intents to production-ready Kubernetes configurations into a dual-encoder transformer architecture that includes 11 best practices checks. The implementation has the following key stages: (1) Data Collection on Kubernetes repositories, (2) Phase 1 Baseline implementation feedback system, (3) Phase 2 Enhanced Dual-Encoder architecture feedback system and (4) Comparative evaluation studies.

2 System Requirements and Tools

Table 1 lists all required tools and their versions.

Table 1: Tools and Version Requirements

Tool	**Version**
Google Colab	Pro (GPU: T4)
Python	3.11.x
PyTorch	2.1.0+
Transformers (HuggingFace)	4.35.0+
Datasets	2.14.0+
evaluate	0.4.0+
codebleu	2.3.1+
PyYAML	6.0+

2.1 Google Colab Environment

Colab Pro is recommended for Phase 2 training (enhanced model requires 15GB+ GPU memory).

3 Repository Setup

3.1 Clone Repository

```
git clone https://github.com/[username]/ravali-thesis.git
cd ravali-thesis
```

3.2 Directory Structure

The repository is organized as follows:

```
ravali-thesis/
        data/
              raw/                        # Kubernetes manifests
              real_manifests/             # Production configs
              training_dataset.json       # Processed training data
              train/                      # Training split
              val/                        # Validation split
              test/                       # Test split
        feedback_system/
              test_results/               # Feedback evaluation outputs
        final_model_v3/
              model.safetensors           # Trained model weights
              config.json                 # Model configuration
              tokenizer.json              # Tokenizer configuration
        results/                        # Evaluation results
        T5_Phase1_Feedback_System_v2_modified.ipynb
        T5_Phase2_Enhanced_DualEncoder_v5_SupervisedOnly_codebleu.ipynb
        T5_Phase2_Enhanced_with_Feedback_System.ipynb
        T5_Quality_Levels_Validation_Study.ipynb
```

4 Data Collection

4.1 Data Sources

The training dataset was collected from three sources:

1. **Kubernetes GitHub Repository** (`kubernetes/examples`)
2. **Helm Charts** (Bitnami, ArtifactHub)
3. **CNCF Projects** (Istio, ArgoCD, Knative)

4.2 Data Collection Process

```
# Clone Kubernetes examples
git clone https://github.com/kubernetes/examples.git
cd examples
find . -name "*.yaml" -o -name "*.yml" > k8s_files.txt
```

4.3 Data Processing

The collected YAML files are processed to create intent-manifest pairs.

Intent statements are generated using templates:

```
# Example intent generation
"Deploy {app_name} with {replicas} replicas and {resource} resources"
```

The final dataset contains 3,456 intent-manifest pairs saved in `data/training_dataset.json`.

5 Phase 1: Baseline Implementation with Feedback

5.1 Baseline Model Training

Notebook: `T5Baseline_Code_v6_Final.ipynb`

1. Mount Google Drive for data access:

   ```
   from google.colab import drive
   drive.mount('/content/drive')
   ```

2. Upload `training_dataset.json` to Google Drive

   - Load and preprocess dataset
   - Initialize CodeT5-base model
   - Configure training parameters
   - Train model with validation
   - Save model checkpoints

3. Expected training time: 2-3 hours on Colab T4 GPU

5.2 Phase 1 Baseline Feedback System Integration

Notebook: `T5_Phase1_Feedback_System_v2_modified.ipynb`

1. Load the trained baseline model from Phase 1
2. Initialize the actionable feedback mechanism with 11 best practices checks:
3. Generate YAML configurations for test cases
4. Analyze each configuration:

   ```
   result = feedback_gen.analyze_yaml(generated_yaml)
   # Returns: BP%, Quality Score, Issues, Remediation
   ```

5. Review outputs in `feedback_system/`

6 Phase 2: Enhanced Dual-Encoder Model

Notebook: `T5_Phase2_Enhanced_DualEncoder_v5_SupervisedOnly_codebleu.ipynb`

This is the main contribution of the research.

6.1 Architecture Overview

The enhanced model uses:

- **Intent Encoder**: BERT-style encoder for natural language understanding
- **Pattern Encoder**: CodeT5 encoder for Kubernetes syntax patterns
- **Fusion Network**: Cross-attention mechanism to combine representations
- **Unified Decoder**: CodeT5 decoder for YAML generation

6.2 Training Procedure

1. Mount Google Drive and upload dataset
2. Training configuration:
    - Batch size: 8
    - Learning rate: 3e-5
    - Epochs: 15
    - Optimizer: AdamW
    - Scheduler: Linear warmup + decay
3. Expected training time: 4-6 hours on V100 GPU
4. Model saves to `final_model_v3/`

7 Comparative Evaluation Studies

7.1 Feedback System Evaluation

Notebook: `T5_Phase2_Enhanced_with_Feedback_System.ipynb`

This notebook compares baseline and enhanced models using the actionable feedback system.

1. Load both models (baseline from Phase 1, enhanced from Phase 2)
2. Define 15 test cases (Simple, Medium, Complex)
3. Generate configurations with both models for identical inputs
4. Analyze with feedback system:

```
baseline_result = feedback_gen.analyze_yaml(baseline_yaml)
enhanced_result = feedback_gen.analyze_yaml(enhanced_yaml)
```

5. Compare metrics:
    - YAML Validity Rate
    - Mean BP Compliance

- Mean Quality Score
- Production-Ready Configs
- Critical/High Issues

6. Run statistical tests (paired t-test, Cohen's d)

7.2 Quality Levels Validation Study

Notebook: `T5_Quality_Levels_Validation_Study.ipynb`

This study evaluates model performance across different intent complexity levels.

1. Define test cases in 4 categories:
   - GOOD: Simple, clear intents
   - NEARLY_GOOD: Moderate complexity
   - BAD: Complex requirements
   - WORSE: Vague/edge cases
2. Test both models on all 8 test cases
3. Calculate quality scores for each category
4. Perform statistical analysis:

```
t_stat, p_value = stats.ttest_rel(enhanced_scores, baseline_scores)
cohens_d = np.mean(diff) / np.std(diff)
```

5. Expected results: Enhanced model shows 29.6 point improvement in quality scores (p=0.0202, d=1.131)

8 Verification and Testing

8.1 Model Verification

Verify the trained model loads correctly:

```
from transformers import AutoTokenizer, T5ForConditionalGeneration

# Load model
model_path = "/content/final_model_v3/"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = T5ForConditionalGeneration.from_pretrained(model_path)

# Test generation
intent = "Deploy nginx with 2 replicas"
inputs = tokenizer(intent, return_tensors="pt", max_length=512)
outputs = model.generate(**inputs, max_length=1024)
yaml_output = tokenizer.decode(outputs[0], skip_special_tokens=True)

print(yaml_output)
```

8.2 Feedback System Testing

Test the feedback analyzer:

```
from src.feedback_generator import FeedbackGenerator

analyzer = FeedbackGenerator()

# Test with sample YAML
sample_yaml = """
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.19
"""

result = analyzer.analyze_yaml(sample_yaml)
print(f"BP%: {result['bp_score']}")
print(f"Quality: {result['quality_score']}")
print(f"Issues: {len(result['issues'])}")
```

8.3 Expected Test Outputs

- Model should generate syntactically valid YAML
- BP% should be 50-90% depending on intent complexity
- Quality scores should range 40-95/100
- Readiness tiers: PRODUCTION_READY (BP85%), DEPLOYMENT_READY (BP75%), NEEDS_WORK (BP75%)

8.4 Key Performance Indicators

Verify that the implementation matches these benchmarks:

Table 2: Expected Performance Benchmarks

Metric	**Baseline**	**Enhanced**
CodeBLEU	30-50%	70%
YAML Validity	50-60%	100%
Mean BP%	55-60%	75-80%
Quality Score	65	80
Production-Ready	0%	50%