**Proposed Title of Thesis**

# Designing a Pipeline for Natural Language Driven Network Configuration using LLMs and Agentic AI

## MR603 Thesis 1

## Candidature Confirmation Report

Student Name: Chandra Shekhar Dhungana

Student ID: MIT251617

Cluster Project Title: AI-Driven Secure and Intelligent Networked Systems

Cluster Lead Name: Dr Mahdi Saki

Supervisor Name: Dr Farhana Afroz

**School of IT and Engineering**

Trimester 2, 2026

Date of Submission of Report: 05 October 2026

## Acknowledgement

I would like to express my sincere gratitude to my Cluster Lead, Dr Mahdi Saki, and my Supervisor, Dr Farhana Afroz, for their guidance, constructive feedback and support throughout the development of this research. Their advice has been invaluable in refining the research direction, methodology and overall quality of this thesis.

I am also grateful to the School of IT and Engineering at the Melbourne Institute of Technology for providing the academic environment and resources that supported the development of this research.

Signature of Student:

Date of Submission of Report: 05 October 2026

## Declaration

I declare that this thesis is my own work, except where sources and contributions have been appropriately acknowledged.

AI Use Declaration**:** AI tools were used to support research planning, academic language refinement, and code development and debugging. All AI-assisted content was critically reviewed, and responsibility for the research, analysis, interpretation and conclusions remains my own.

Signature of Student:

Date of Submission of Report: 05 October 2026

## Abstract

Network configuration is a fundamental part of network management, but translating operational requirements into correct device-level configurations remains difficult because configurations are precise, vendor-specific and sensitive to small errors. Large Language Models (LLMs) provide a promising way to translate natural-language intents into configurations, yet existing studies use different models, datasets, vendors and evaluation methods, making it difficult to determine which techniques actually improve accuracy and whether their benefits justify the computational cost.

This research investigates how an open-weight LLM-based pipeline can generate syntactically and logically accurate Cisco IOS and Juniper Junos configurations while remaining computationally efficient. The study uses a controlled experimental design to evaluate fine-tuning and Retrieval-Augmented Generation (RAG), individually and in combination, across selected Qwen and Llama models. Selected models will then be evaluated within an agentic workflow that uses external validation and bounded iterative correction. Evaluation will combine reference-based measures with syntactic and logical accuracy, latency and computational-cost measurements, using a separate multi-vendor benchmark for consistent comparison.

The initial findings show that the required dataset, fine-tuning process and evaluation environment are available to carry out the proposed research. They also show that broader training can affect Qwen and Llama differently, supporting the need for controlled comparison. These findings demonstrate the feasibility of the research rather than provide final answers to the research questions. The remaining experiments will identify which techniques improve accuracy and efficiency and will use this evidence to design the final network-configuration pipeline.

## Glossary and Abbreviations

| Abbreviation | Full Form / Meaning |
|---|---|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| BERT | Bidirectional Encoder Representations from Transformers |
| BGP | Border Gateway Protocol |
| Bi-LSTM | Bidirectional Long Short-Term Memory |
| BLEU | Bilingual Evaluation Understudy |
| BP | Brevity Penalty |
| CI | Confidence Interval |
| CLI | Command-Line Interface |
| CPU | Central Processing Unit |
| CSV | Comma-Separated Values |
| CUDA | Compute Unified Device Architecture |
| DHCP | Dynamic Host Configuration Protocol |
| EM | Exact Match |
| F1 | F1 score (harmonic mean of precision and recall) |
| FAISS | Facebook AI Similarity Search |
| FN | False Negative |
| FP | False Positive |
| FT | Fine-Tuning |
| GB | Gigabyte |
| GNS3 | Graphical Network Simulator-3 |
| GPT | Generative Pre-trained Transformer |
| GPU | Graphics Processing Unit |
| HRAG | Hierarchical Retrieval-Augmented Generation |
| IBN | Intent-Based Networking |
| IOS | Internetwork Operating System |
| IP | Internet Protocol |
| JSON | JavaScript Object Notation |
| JSONL | JSON Lines |

| Abbreviation | Full Form / Meaning |
|---|---|
| LLM | Large Language Model |
| LoRA | Low-Rank Adaptation |
| MRQ | Main Research Question |
| NF4 | 4-bit NormalFloat |
| NIT | Network Intent Translation |
| NLP | Natural Language Processing |
| NMS | Network Management System |
| NR | Not Reported |
| OSPF | Open Shortest Path First |
| PEFT | Parameter-Efficient Fine-Tuning |
| QLoRA | Quantized Low-Rank Adaptation |
| RAG | Retrieval-Augmented Generation |
| RDF | Resource Description Framework |
| RO | Research Objective |
| ROUGE | Recall-Oriented Understudy for Gisting Evaluation |
| ROUGE-L | ROUGE Longest Common Subsequence |
| SDN | Software-Defined Networking |
| SRQ | Sub-Research Question |
| TP | True Positive |
| TRL | Transformer Reinforcement Learning |
| USD | United States Dollar |
| VLAN | Virtual Local Area Network |
| VPN | Virtual Private Network |
| VoIP | Voice over Internet Protocol |
| YAML | YAML Ain't Markup Language |

## Table of Figures

Figure 1.1. Conceptual Overview of the Proposed Natural-Language-Driven Network Configuration Pipeline

Figure 2.1. Structure of the Literature Review

Figure 2.2. Fine-Tuning and RAG Integration for Network Configuration Generation

Figure 2.3. Validation Levels for LLM-Generated Network Configurations

Figure 4.1. Research Methodology Framework

Figure 4.2. Dataset Development

Figure 4.3. Proposed Experimental Framework for LLM-Based Network Configuration

Figure 7.1. Cisco-Specific Fine-Tuning Workflow

Figure 7.2. Multi-Vendor and Multi-Task Fine-Tuning Workflow

Figure 7.3. Exact-Match Transitions Following Multi-Task Fine-Tuning

Figure 7.4. Effect of Three-Shot Prompting on Line-F1

Figure 7.5. Case-Level Effects of Multi-Task Fine-Tuning

## Table of Tables

Table 2.1. Vendor Coverage by Study

Table 2.2. Pipeline Component Coverage

Table 2.3. Key Studies, Remaining Gaps and Research Response

Table 3.1. Research Question and Objective Alignment

Table 4.1. Dataset Sources

Table 7.1. Fine-Tuning Settings

Table 7.2. Training Resources and Cost

Table 7.3. Three-Shot NetConfRAG Results

Table 8.1. Current Trimester Timeline

Table 8.2. Experimental Timeline

## Table of Contents

Acknowledgement ..... 2
Declaration ..... 3
Abstract ..... 4
Glossary and Abbreviations ..... 5
Table of Figures ..... 7
Table of Tables ..... 8
Chapter 1: Introduction ..... 14
1.1 Research Context and Motivation ..... 14
1.2 Challenges and Current Research Approaches ..... 14
1.3 Evaluation Motivation and Study Focus ..... 16
1.4 Background ..... 17
1.5 Thesis Structure ..... 18
Chapter 2: Literature Review ..... 19
2.1 Introduction ..... 19
2.2 Conventional Network Configuration Automation ..... 20
2.2.1 Requirement-to-Configuration Translation Methods ..... 20
2.2.2 Strengths and Limitations of Conventional Approaches ..... 22
2.2.3 Formal Synthesis and Verification ..... 22
2.3 LLM-Based Network Configuration Generation ..... 23
2.3.1 Intent Interpretation and Topology-Aware Configuration Generation ..... 23
2.3.2 Model Capability and Prompting Strategies ..... 24
2.3.3 Structured Output Generation ..... 25
2.4 Domain Knowledge Integration ..... 26
2.4.1 Foundations of Parameter-Efficient Fine-Tuning and Retrieval-Augmented Generation ..... 26
2.4.2 Domain-Specific Fine-Tuning ..... 27
2.4.3 Retrieval-Augmented Generation Using Vendor Documentation ..... 29
2.4.4 Combined Fine-Tuning and Retrieval for Multi-Vendor Configuration ..... 30
2.5 Validation and Iterative Correction ..... 32
2.5.1 Syntactic and Logical Validation ..... 32
2.5.2 Tool-Assisted and Agentic Correction ..... 34
2.6 Evaluation of Correctness and Efficiency ..... 36
2.6.1 Benchmarks and Evaluation Metrics ..... 36
2.6.2 Computational Cost and Latency ..... 38

2.6.3 Evidence from Operational Network Environments ..... 39
2.7 Research Gap ..... 42
Chapter 3: Aim and Objectives of Research ..... 43
3.1 Problem Statement ..... 43
3.2 Research Aim ..... 44
3.3 Research Questions ..... 44
3.3.1 Main Research Question (MRQ) ..... 44
3.3.2 Sub-Research Question 1 (SRQ1) ..... 45
3.3.3 Sub-Research Question 2 (SRQ2) ..... 45
3.4 Research Objectives ..... 45
3.5 Alignment of Research Questions and Objectives ..... 45
3.6 Research Originality and Expected Contributions ..... 46
Chapter 4: Research Design and Proposed Method ..... 47
4.1 Research Approach ..... 47
4.2 Dataset Development and Preparation ..... 49
4.2.1 Data Sources and Acquisition ..... 49
4.2.2 Dataset Content and Task Coverage ..... 50
4.2.3 Source-Specific Cleaning and Quality Control ..... 50
4.2.4 Targeted Juniper Data Augmentation ..... 51
4.2.5 Dataset Integration and Training Format ..... 51
4.2.6 Partitioning and Leakage Checks ..... 52
4.3 Model Selection and Baseline Establishment ..... 52
4.4 Model Fine-Tuning and Optimization ..... 53
4.4.1 Model Adaptation Procedure ..... 53
4.4.2 Controlled Fine-Tuning Experiments ..... 53
4.4.3 Validation-Guided Optimization ..... 53
4.5 Retrieval-Augmented Generation ..... 54
4.6 Agentic Validation and Feedback ..... 54
4.7 Benchmark Development and Evaluation Framework ..... 55
4.8 Quantitative Analysis and Pipeline Selection ..... 56
4.9 Evaluation Metrics ..... 57
4.9.1 Reference-Based Metrics ..... 57
4.9.2 Configuration Validity Metrics ..... 58
4.9.3 Computational and Cost Metrics ..... 59

Chapter 5: Research Requirements Analysis and Specification ..... 59
5.1 Language Models ..... 59
5.1.1 Initial Experimental Models ..... 59
5.1.2 Extended Model Candidates ..... 59
5.2 Dataset Preparation and Augmentation Tools ..... 59
5.3 Fine-Tuning Tools and Libraries ..... 60
5.3.1 PyTorch and Hugging Face Framework ..... 60
5.3.2 Parameter-Efficient Fine-Tuning ..... 60
5.3.3 Training Outputs and Model Storage ..... 60
5.4 Computing and Development Environment ..... 61
5.5 Retrieval-Augmented Generation Tools ..... 61
5.6 Automation and Agentic Workflow Tools ..... 61
5.7 Network Verification and Simulation Tools ..... 62
5.7.1 Batfish and PyBatfish ..... 62
5.7.2 GNS3 as a Possible Complement ..... 62
5.8 Evaluation and Analysis Tools ..... 62
5.9 Reproducibility and Version Control ..... 62
5.10 Technical Environment Summary ..... 63
Chapter 6: Ethical Considerations ..... 63
Chapter 7: Initial Findings ..... 64
7.1 Scope of the Initial Experimental Investigation ..... 64
7.2 Experimental Progression and Training Data ..... 64
7.2.1 Cisco-Specific Baseline Experiment ..... 64
7.2.2 Multi-Vendor and Multi-Task Extension ..... 65
7.3 Fine-Tuning Configuration and Computational Resources ..... 66
7.3.1 Model Adaptation Setup ..... 66
7.3.2 Computational Cost and Resource Usage ..... 67
7.4 Evaluation Design and Metrics ..... 68
7.4.1 External Benchmark and Prompting Conditions ..... 68
7.4.2 Evaluation Metrics ..... 68
7.5 Experimental Results ..... 69
7.5.1 Overall Benchmark Performance ..... 69
7.5.2 Qwen Performance ..... 70
7.5.3 Llama Performance ..... 71

7.5.4 Zero-Shot and Three-Shot Behavior ..... 71
7.6 Principal Empirical Observation and Interpretation ..... 72
7.6.1 Improvement Under Broader Training Supervision ..... 72
7.6.2 Potential Cross-Vendor and Cross-Task Transfer ..... 74
7.6.3 Alternative Explanations and Need for Ablation ..... 74
7.7 Limitations and Interpretation Boundaries ..... 74
7.8 Implications and Next Experimental Steps ..... 74
Chapter 8: Timeline for Research ..... 75
8.1 Current Trimester ..... 75
8.2 Upcoming Experimental Trimester ..... 76
Chapter 9: Conclusion, Limitations and Future Work ..... 77
9.1 Conclusion and Progress to Date ..... 77
9.2 Limitations and Research Constraints ..... 78
9.3 Future work ..... 80
Chapter 10: Plans for Future Publication ..... 81
References ..... 82
Appendices ..... 87
Appendix A: Reproducibility and Research Artefacts ..... 87
A.1 Research Code Repository ..... 87
A.2 Repository Structure and Implemented Scope ..... 87
A.3 Reproducibility Record ..... 88
A.4 Hugging Face Adapter Artefacts ..... 89
Appendix B: Dataset and Implemented Experimental Record ..... 89
B.1 Prepared Dataset Composition ..... 89
B.2 Supervised Fine-Tuning Record Format ..... 90
B.3 Partitioning and Leakage Controls ..... 90
B.4 Implemented QLoRA Configuration ..... 91
B.5 External Evaluation Protocol ..... 92
Appendix C: Supplementary Preliminary Results ..... 92
C.1 Line-Level Error Profile ..... 93
C.2 Sensitivity to Configuration Length ..... 94
C.3 Distribution of Per-Case Line-F1 ..... 95
C.4 Accuracy-Latency Trade-Off ..... 96
Appendix D: Key Implementation Code Excerpts ..... 97

D.1 Group-aware dataset partitioning ..... 98
D.2 Fuzzy leakage screening ..... 99
D.3 Typed Juniper augmentation value generation ..... 100
D.4 Centralized QLoRA experiment configuration ..... 101
D.5 LoRA adapter construction ..... 102
D.6 Reproducibility Interpretation ..... 102
Appendix E: Diagram Development Artefacts ..... 103

# Chapter 1: Introduction

## 1.1 Research Context and Motivation

Network automation is essential because manual network configuration is complex, costly, and prone to human error, which can lead to service disruptions and increased operational effort [1].

Intent-Based Networking (IBN) plays a crucial role in enabling autonomous networks by allowing goals and constraints to be specified at a higher level to the Network Management System (NMS) [2]. It introduces the notion of an “Intent,” which represents an abstract operational goal provided as input to the NMS. The NMS then generates the necessary low-level configurations required to fulfill that intent [3].

Before Large Language Models (LLMs), IBN relied on approaches such as policy-based automation, predefined JavaScript Object Notation (JSON) or YAML Ain't Markup Language (YAML) intent structures, and traditional natural language processing techniques to interpret and translate network requirements. However, these approaches often require specialized knowledge and strict input formats. LLMs offer a promising improvement by allowing users to express network requirements in natural language, simplifying intent interpretation and translation while reducing the need for detailed technical knowledge [4].

## 1.2 Challenges and Current Research Approaches

**Problems using LLMs for network configuration**

- Hallucination and reliability: LLMs may generate syntactically plausible but incorrect, unsupported, or functionally invalid network configurations. Studies have demonstrated errors in configuration syntax, topology, and semantics, making direct deployment without verification risky [5], [6].
- Limited domain knowledge and data: General-purpose LLMs may lack sufficient network- and vendor-specific knowledge, while high-quality network-specific training data remain scarce [5], [7].

These problems have been extensively discussed in recent research, but there is currently no universally reliable solution. LLMs generate outputs probabilistically, meaning hallucinated or incorrect outputs remain an important reliability concern rather than being completely

avoidable through prompting alone [5]. This is particularly significant for network configuration, where precise syntax and semantics are required and relatively small configuration errors may lead to service interruption, incorrect routing, or security problems. Mondal et al., for example, found that GPT-4 could produce promising router configurations while still containing substantial topology, syntax, and semantic errors [6].

Fine-tuning addresses part of this problem by further training a pretrained language model using domain-specific data so that the model becomes better adapted to a particular task or knowledge domain. Several studies have explored this approach for networking. PreConfig constructs a specialized network-configuration corpus and fine-tunes a programming language model to address the lack of domain knowledge in general-purpose models [7]. More recently, SLM_netconfig uses Low-Rank Adaptation (LoRA) based fine-tuning and a domain-specific configuration dataset to specialize a smaller model for intent-to-configuration generation [8]. These studies demonstrate that fine-tuning can improve domain-specific configuration performance, although it depends heavily on the availability and quality of specialized training data, which remains a recognized limitation in networking [9].

Another approach is Retrieval-Augmented Generation (RAG). Rather than relying only on knowledge stored within the model parameters, RAG retrieves relevant external information, typically from an indexed collection of technical documentation, and supplies this information to the LLM during generation. This is particularly relevant to network configuration because vendor commands and syntax are highly specialized and can change over time. Ouni et al. evaluate hierarchical RAG over networking documentation and report better performance than an LLM-only baseline [9]. Similarly, ConfGen retrieves vendor-specific syntax information before generating multi-vendor configurations for Huawei, Cisco, Nokia, and Juniper [10]. These studies suggest that external grounding can reduce dependence on the model's internal knowledge, although retrieval quality and generation reliability remain important factors.

Research on LLM-based network configuration has expanded rapidly in recent years, but the field remains relatively young. A recent survey specifically identifies scarcity of network-specific training data, hallucination, and domain-adaptation difficulties as continuing challenges [5]. Moreover, research on Generative Artificial Intelligence (AI) for IBN has been

described as being concentrated heavily on intent translation, with substantial research gaps remaining across other stages of the IBN lifecycle [11].

This movement is also visible in industry practice. Cisco now describes agentic AI frameworks that perform tasks such as configuration-drift remediation, troubleshooting, validation, and autonomous network operations, while HPE Juniper describes LLM- and agentic-AI-based interfaces for conversational network operation, monitoring, troubleshooting, and automation [12], [13].

### 1.3 Evaluation Motivation and Study Focus

Existing academic studies also use substantially different models, datasets, prompts, vendors, tasks, and validation environments. Consequently, comparing headline accuracy values between papers does not necessarily provide a controlled comparison. For example, comparing a fine-tuned smaller open model against an unmodified proprietary model can demonstrate practical performance differences, but it does not isolate the effect of fine-tuning, because model family, model size, training data, and other factors change simultaneously. This broader evaluation problem has been recognized by NETLLMBENCH, which states that a systematic method for evaluating LLM performance in network configuration was lacking and therefore introduces a common benchmarking and network-emulation framework [14].

Another challenge is the rapid evolution of the underlying models. Recent studies evaluating frontier models such as GPT-4 and GPT-5 still report imperfect performance across network configuration translation tasks, showing that newer models have not eliminated configuration errors. At the same time, the use of large proprietary models can introduce substantial inference and evaluation cost [14], [15]. Results obtained with a particular model can become less representative as newer models are released. Furthermore, although validation and agentic approaches are beginning to appear, they are not yet consistently incorporated across network-configuration studies. CoSynth combines GPT-4 with configuration verifiers and feeds verifier errors back to the model [6]. NETLLMBENCH uses closed-loop network emulation [14]. More recent work has also explored agent-based configuration systems and multi-agent network-management architectures [8], [16]. Therefore, closed-loop and agentic validation are active research directions rather than a standard component of all LLM-based configuration systems.

Within this context, this research focuses on multi-vendor network configuration using Cisco IOS and Juniper Junos. Cisco and Juniper were selected because they are among the major vendors in the global routing and network infrastructure market, making them relevant platforms for evaluating LLM-based network configuration across different vendor environments.

Our research will focus on current LLM-based network-configuration techniques and evaluate them under a more consistent experimental environment. Multiple models will be assessed using the same tasks, datasets, evaluation criteria, and network conditions where possible. Rather than examining only whether one model outperforms another, the experiments will investigate which factors produce the performance change. This includes comparing the same model before and after fine-tuning, evaluating generation with and without RAG, comparing zero-shot, one-shot and few-shot prompting, and examining whether validation and iterative correction improve configuration correctness. This design will allow the contribution of each technique to be analyzed more directly while also examining computational cost, reliability and latency.

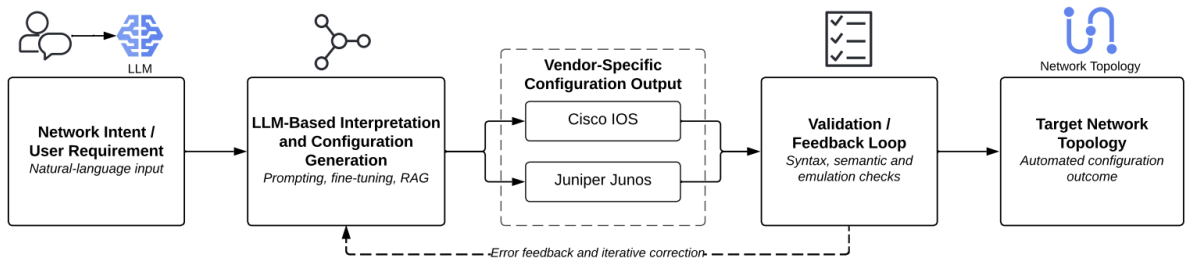



Figure 1.1. Conceptual overview of the proposed natural-language-driven network configuration pipeline

## 1.4 Background

Large Language Models (LLMs) are language models trained on large collections of natural-language and code data, enabling them to interpret varied instructions and generate structured or technical outputs. Modern LLMs are predominantly based on the Transformer architecture [17], which supports the contextual processing of relationships within an input sequence. For network automation, their main significance is their ability to interpret requirements expressed in relatively unrestricted natural language rather than requiring users to follow a predefined intent language or rigid input schema. This creates the possibility

of using an LLM as an interface between human network requirements and lower-level representations required by network-management systems. Recent studies have therefore investigated LLMs for translating network intents into formal specifications, API operations and device configurations. Tu, Nam and Hong examine LLM-based intent translation using fine-tuning and in-context learning [18], while NetConfEval evaluates LLMs across formal-representation, function-call and low-level configuration tasks [1]. These capabilities provide the technical foundation for investigating natural-language-driven network configuration in this research.

### 1.5 Thesis Structure

The remainder of this thesis is organized as follows.

Chapter 2 critically reviews the relevant literature on network automation, LLM-based configuration generation, fine-tuning, RAG, validation, and agentic correction, and establishes the research gap addressed by this study.

Chapter 3 defines the research problem, aim, research questions, and objectives that guide the investigation.

Chapter 4 presents the research methodology and experimental design, including dataset development, model fine-tuning, RAG, validation, and the evaluation framework.

Chapter 5 specifies the technical requirements and research environment, including the selected models, computational resources, software tools, verification mechanisms, and reproducibility procedures.

Chapter 6 addresses the ethical and research-integrity considerations associated with the use of datasets, technical documentation, AI models, and simulated network environments.

Chapter 7 presents the preliminary experimental work and findings completed to date and discusses their implications for the subsequent stages of the research.

Chapter 8 outlines the research timeline and the planned sequence of activities for completing the remaining experimental work.

Chapter 9 concludes the report by summarizing the research progress, identifying the principal limitations and constraints, and outlining the remaining work required to complete the study.

Chapter 10 outlines the planned publication strategy arising from the research.

## Chapter 2: Literature Review

### 2.1 Introduction

Natural-language-driven network configuration raises a central question: under what conditions can an operator's requirements be translated into technically correct configurations at a practical computational cost? Producing plausible commands is only one part of this problem. A dependable pipeline must correctly interpret the intended requirement, respect vendor and topology constraints, generate appropriate configurations, and identify errors before the output is accepted. Existing studies address different parts of this process, including intent interpretation, configuration synthesis, domain adaptation, retrieval, verification, repair and operational assistance [1], [14], [19].

This review therefore examines how networking knowledge and correctness are introduced at different stages of the configuration pipeline. It first considers network automation approaches that do not rely on LLMs, establishing the mechanisms and guarantees available before generative models. It then examines LLM-based configuration generation, domain-specific fine-tuning, retrieval-augmented generation (RAG), structured generation, and tool-assisted validation and correction. Particular attention is given to how these approaches are evaluated, because reported performance depends strongly on the task, vendor, validation method and metric used. Cisco IOS and Juniper Junos provide the primary target environments for this study. The review concludes by identifying unresolved questions concerning the individual and combined effects of fine-tuning, RAG and iterative validation, which directly motivate the experimental design of this research.

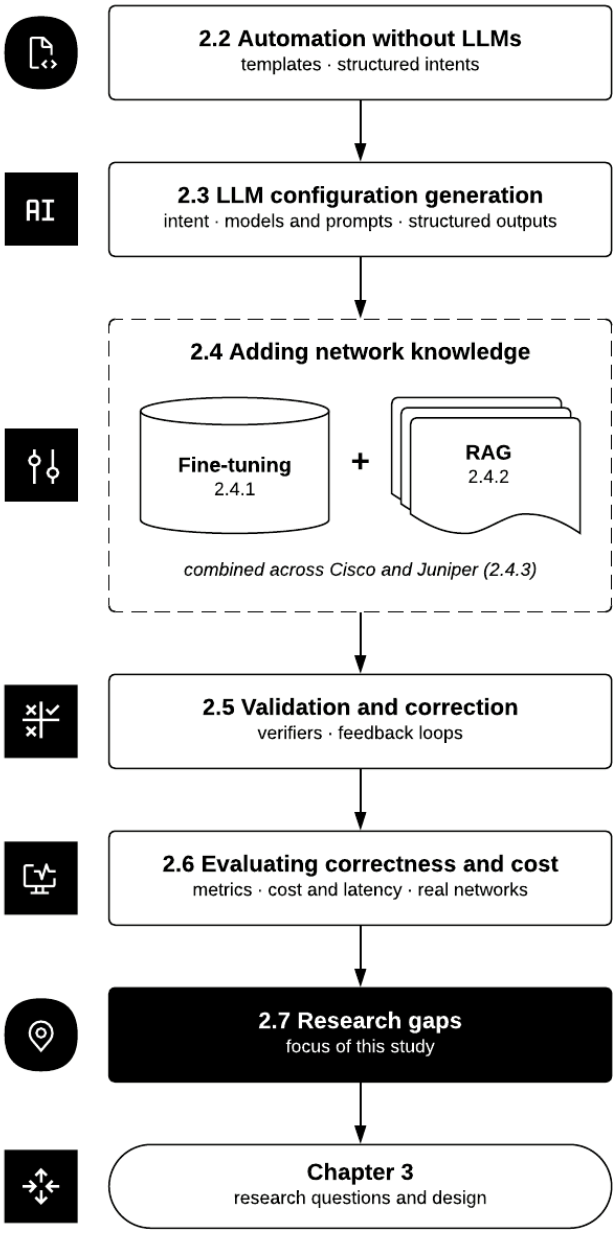



Figure 2.1. Structure of the Literature Review

## 2.2 Conventional Network Configuration Automation

### 2.2.1 Requirement-to-Configuration Translation Methods

Automation without LLMs provides several ways of locating networking expertise outside a generative model. In template-based systems, experts specify configuration rules and software instantiate them using operational data. PRESTO embodies this approach through reusable configlets written close to device-native configuration languages. A compiler combines these fragments with information from external systems to produce configurations

for requested services [20]. Its input is consequently more structured than unrestricted natural-language instruction: service requirements must be represented in data models that the templates can consume. The automation removes repeated construction work after engineers have encoded the service logic.

PRESTO also shows why templates should not be dismissed as simple text substitution. Supporting a new service involves identifying provisioning information, understanding available data sources, codifying engineering rules and designing the surrounding workflow. Its Virtual Private Network (VPN) and Voice over Internet Protocol (VoIP) applications use different compositions and output forms, including complete configurations and configuration snippets [20]. Expertise is distributed across the templates, databases and provisioning process. This distribution makes the source of configuration decisions inspectable, although the system remains dependent on accurate operational inputs and maintained engineering knowledge.

Natural-language interfaces can precede this constrained translation without using an LLM. The extended LUMI architecture extracts entities through supervised bidirectional Long Short-Term Memory (Bi-LSTM) and conditional random field models, assembles an intermediate Nile intent and uses a Random Forest classifier to detect ambiguity between intents. Missing information and conflicting interpretations are referred to the operator [21]. The architecture treats language interpretation and deployment as separate responsibilities. Its structured language makes supported operations explicit, while human confirmation supplies information that the automated interpretation cannot safely resolve.

NAIL offers another division of responsibility for programmable switches. Its natural-language processing, management objects and transpiler convert supported intent descriptions into data-plane programs, with an implementation centered on P4. Management operations and monitoring connect translation to the affected devices [22]. These mechanisms differ from PRESTO's device-native configlets, but both place substantial domain knowledge in explicit software structures. The relevant comparison with LLMs is where interpretation and translation occur, not whether automation exists at all.

Other non-LLM approaches distribute the same problem differently. iNDIRA uses conventional natural language processing (NLP) and semantic Resource Description Framework (RDF) representations to translate natural-language requests into scientific-

network provisioning actions [23]. KID combines a knowledge graph with a digital twin to support intent translation and network-state reasoning [24]. INCS represents topology, configuration sketches and requirements as a graph, then combines graph learning with deep reinforcement learning for network-wide synthesis [25]. Yu et al. place translation within a broader open-source IBN lifecycle that includes orchestration, assurance and corrective actions [26]. Ribeiro et al. work in the reverse direction, deterministically extracting security intents from existing configurations into the Nile language, illustrating the value of explicit semantic mappings [27].

### 2.2.2 Strengths and Limitations of Conventional Approaches

Explicit translation structures provide useful control by narrowing the space in which configuration decisions are made. PRESTO demonstrates substantial service provisioning through expert-authored generation [20]. Its strongest evidence concerns new configurations, with more limited support for modifying existing installations. The comparison with LLM generation should preserve the starting-state assumptions of each task.

The constrained language and confirmation stages in LUMI improve transparency, but supported entities and operations still determine the requirements it can represent. Its ambiguity classifier also remains a learned component, not itself a formal guarantee that all conflicts have been removed [21]. Similarly, NAIL's deployment and monitoring mechanisms demonstrate useful control within programmable data planes; they do not establish general Cisco IOS or Juniper Junos translation [22].

The comparison suggests that a flexible language interface should retain explicit requirements, traceable translation and appropriate validation. An LLM might reduce the effort needed to map varied wording onto configuration operations, but it can relocate engineering work into data preparation, documentation retrieval and checking. PRESTO makes this pre-existing knowledge-engineering burden visible [20].

### 2.2.3 Formal Synthesis and Verification

Formal synthesis provides a useful baseline for what can be guaranteed when requirements are expressed in constrained forms. Propane compiles high-level path-policy constraints into router-level Border Gateway Protocol (BGP) configurations and guarantees that the compiled policy is preserved under all combinations of failures [28]. NetComplete instead starts from

partial configurations with explicit holes and autocompletes static-route, Open Shortest Path First (OSPF) and BGP parameters, scaling to networks of up to 200 routers in its evaluation [29]. These systems show that strong correctness claims are achievable when the input specification, supported protocols and synthesis space are explicitly bounded.

Verification work complements this synthesis line by checking the semantics of existing configurations. Minesweeper translates router configurations and routing-protocol interactions into logical constraints for properties including reachability, isolation, waypointing and fault tolerance; its evaluation checked 152 real networks and found 120 violations [30]. Campion compares router configurations for behavioral equivalence and localizes differences to relevant configuration components, identifying four bugs among 30 proposed router replacements in a production cloud network [31]. Together, these systems establish a pre-LLM tradition of evaluating configuration semantics against explicit properties, not textual resemblance, a distinction later LLM pipelines retain when they use external synthesis and verification tools.

## 2.3 LLM-Based Network Configuration Generation

### 2.3.1 Intent Interpretation and Topology-Aware Configuration Generation

Intent interpretation is distinct from producing a configuration that realizes the interpreted requirement. Tu, Nam and Hong evaluate LLM-based intent translation through fine-tuning and in-context learning, showing that the translation stage can itself be adapted before configuration is applied [18]. NetIntent similarly separates intent translation from later activation and assurance while benchmarking controller-specific JavaScript Object Notation (JSON) generation across four Software-Defined Networking (SDN) platforms [32]. LLM-NetCFG classifies high-level requests, expresses lower-level requirements in JSON and subsequently generates configurations; its reported 92.2% accuracy concerns classification of 90 intents, not successful fulfillment of all requests [33]. An early interpretation error can influence both generation and the choice of validation.

Network structure creates a further dependency. Donadel et al. find that LLM topology reasoning degrades as network complexity increases, with errors including incorrect paths and invented connections [34]. GeNet adds visual topology input and shows that a multimodal model can interpret and modify topology and configuration across ten enterprise-style

scenarios, including low-quality sketches [35]. In fastCEGS, intent and topology instead guide graph-based retrieval of compatible examples while local verification checks attributes against the target topology [36]. These designs make structural context an explicit input and no longer leave it implicit in a prompt.

These findings suggest that completeness of the represented requirement is a prerequisite for meaningful downstream checks. If an interface, policy boundary, or required relationship is not captured during interpretation, verifying only the generated representation may overlook the omission. NetConfEval's separation of formal specification translation from later configuration tasks supports evaluating these stages individually [1]. Intent understanding and configuration correctness should remain distinct evaluation targets.

### 2.3.2 Model Capability and Prompting Strategies

NetLLMBench evaluates several open models on nine tasks in the same small emulated network, providing a model comparison under consistent task, prompt and assessment conditions; Llama3-70B completes them in one iteration, while other models encounter formatting or configuration difficulties [14]. The result is strong benchmark-specific evidence, not a general ranking for complex vendor-specific routing policies.

Prompt demonstrations can also change the apparent capability being measured. NetIntent benchmarks 33 open models and reports that smaller or code-oriented models can remain competitive with much larger models; relevant in-context examples substantially improve some mid-sized models [32]. LLNet likewise shows that examples influence structured-JSON translation alongside input length and processing cost [37]. Mekrache et al. retrieve previously validated intent–output examples for few-shot prompting before validating the generated intermediate intent [3]. A model comparison that changes model size and example selection together would combine distinct effects.

Documentation-assisted generation reveals an additional limit of prompting. In NetConfEval, supplying documentation does not yield a uniform improvement across low-level configuration scenarios. Retrieval can select material that fails to provide the required context, while document organization affects how effectively relevant information reaches the model [1]. The presence of correct information somewhere in the documentation collection is thus different from its availability in the actual prompt. Even availability does not ensure correct application.

Taken together, these comparisons favor conditional conclusions about model suitability. A small model may be adequate when the output vocabulary is constrained, and examples closely match the task; a difficult configuration dependency may remain unresolved despite a larger model or longer prompt. The reviewed experiments support testing these possibilities under matched conditions, without constructing a universal model ranking from values[1], [14], [37].

### 2.3.3 Structured Output Generation

Intent-LLM supplies a documented network application programming interface (API) and asks GPT-4o Mini to generate Python code that composes predefined functions, showing how an intermediate representation changes the task assigned to the LLM [38]. LLNet asks the model to produce JSON describing the requested operation, after which deterministic program logic generates OpenFlow, P4 or eBPF rules [37]. Both approaches constrain the model through an explicit interface and avoid free-form vendor command-line interface (CLI) generation.

Configuration sketches create a different boundary. LCLE extracts workflows, command syntax and view relationships from manuals and uses them to generate sketches [39]. By leaving parameter values outside the evaluated sketch-generation task, it can assess command selection and structural validity separately from complete instantiation. fastCEGS also divides synthesis, but uses retrieved examples and LLM-generated templates before NetComplete [29] resolves symbolic parameters under formal constraints [36]. These approaches constrain generation at different stages: one organizes documentation-derived configuration structure, whereas the other combines template adaptation with subsequent formal completion.

GIA shows why a structured output is valuable only in relation to what its schema expresses and what the next component guarantees. It has an LLM generate intent graphs through Python code and then uses subgraph isomorphism for capability negotiation; its error analysis still records imaginary nodes, links and attributes as intent complexity rises [40]. LCLE's separation of syntax validity from precision and recall makes related structural concerns visible [39]. Structure can expose and constrain model decisions, but it does not make those decisions correct by itself.

## 2.4 Domain Knowledge Integration

### 2.4.1 Foundations of Parameter-Efficient Fine-Tuning and Retrieval-Augmented Generation

The adaptation methods used in network-configuration studies differ mainly in where specialized knowledge enters the pipeline. Full fine-tuning updates a pretrained model's parameters using domain data, but parameter-efficient fine-tuning trains only a small set of parameters, often added alongside the frozen model. LoRA freezes the original weights and learns low-rank update matrices within selected layers, reducing the number of trainable parameters and the memory needed for adaptation [41]. The resulting adapter is small relative to the base model, although preparing a high-quality domain dataset and running training still create one-off computational and engineering costs.

Quantized Low-Rank Adaptation (QLoRA) reduces this training footprint further by keeping the pretrained model in 4-bit quantized form while training LoRA adapters with higher-precision computation where needed [42]. Its importance is practical: larger open-weight models can be adapted on much smaller Graphics Processing Unit (GPU) resources than conventional full fine-tuning would require. Quantization does not supply networking knowledge by itself. The knowledge still comes from the examples used during adaptation, so data provenance, vendor balance and validation remain central to what the adapter learns.

RAG introduces knowledge through a different channel. The original RAG formulation combines a generator with retrieved external passages, allowing generation to condition on information selected at inference time without requiring every fact to be encoded in model parameters [43]. In network configuration, this distinction is useful because vendor manuals contain detailed syntax and platform constraints that may change independently of the base model. Retrieval avoids retraining whenever documentation changes, but it adds indexing, retrieval and context-processing costs to each query and can fail when relevant material is not selected or correctly applied.

Fine-tuning places more preparation cost before deployment and can make recurring generation simpler, whereas RAG shifts part of the knowledge-selection work into inference. Combining them can be useful when recurring configuration patterns are learned through adaptation while platform-specific details are retrieved when needed. The literature reviewed below tests these mechanisms under different datasets, vendors and validation

methods, so their effects must be separated from differences in model family, prompt design and evaluation.

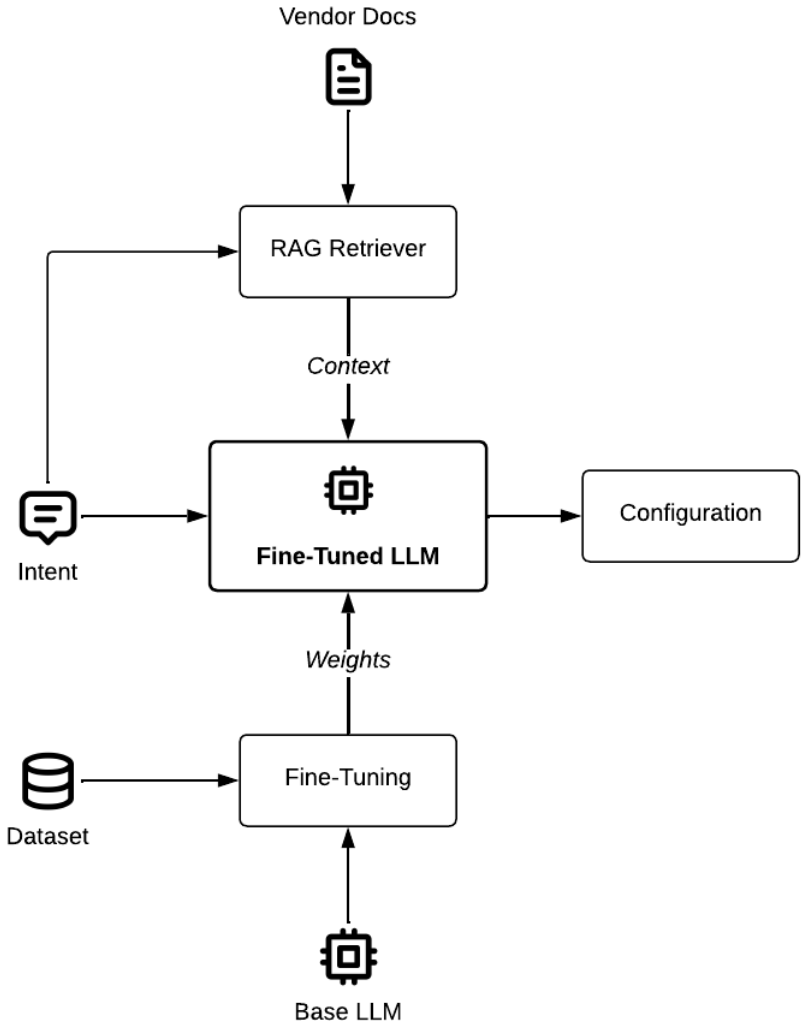



Figure 2.2. Fine-Tuning and RAG Integration for Network Configuration Generation

### 2.4.2 Domain-Specific Fine-Tuning

Domain-specific fine-tuning makes networking knowledge available through the model's learned parameters. PreConfig applies LoRA-based instruction fine-tuning to Qwen2.5-Coder-Instruct 1.5B for generation, analysis, translation and completion across Cisco and Juniper [7]. Its approximately 2.13 million configuration lines quantify corpus volume, not independent intent–configuration cases, so task coverage is more informative than line count alone.

The provenance and technical validation of examples matter as much as their volume. PreConfig uses generated instruction data and describes Campion [31] checks within the

construction of translation supervision [7]. This gives the training data a stronger technical filter than unverified synthetic translation. Dataset validation and evaluation of newly generated configurations nevertheless answer different questions: the former supports supervision quality, while the latter tests whether the learned model produces correct outputs on unseen tasks. PreConfig's reported similarity and configuration-aware metrics primarily support conclusions about its learned configuration tasks.

Lira et al. provide a complementary comparison through LoRA adaptation of Zephyr-7B using Cisco IOS XE documentation. They distinguish requirement-based and question-based training, selecting the latter for their final system [8]. On 150 unseen Cisco tasks, the paper reports 65.3% fully correct syntax and 66.7% fully correct goal accuracy. These are separate assessment categories, not interchangeable estimates of successful deployment. The evaluation is useful precisely because it does not collapse conformity of the command language and satisfaction of the stated goal into one undifferentiated score.

The same study shows clear vendor sensitivity. Performance transfers weakly from the Cisco-adapted model to Huawei, then improves after additional Huawei fine-tuning [8]. Juniper transfer and balanced multi-vendor training remain open questions, and pooled scores could hide these differences.

Mobile-LLaMA provides parallel evidence from 5G network analysis rather than router CLI generation. Kan et al. instruction-fine-tune LLaMA 2 13B on domain data expanded through self-instruction and report stronger network-analysis code generation than their GPT-3.5 comparison [44]. The study supports domain adaptation as a mechanism for specialized networking knowledge, while its packet, routing-analysis and performance tasks remain different from Cisco or Juniper configuration generation.

Dataset construction also changes the evidential value of fine-tuning. AlSamarneh et al. manually develop 1,000 Juniper Junos intent–configuration pairs in NIT, check configurations against vendor manuals and execute them on a physical Juniper EX3300 switch [45]. The prompts include complete, partial and missing parameters, but the dataset targets one switch platform and does not resolve conflicts or validate arbitrary parameter choices. NIT supplies unusually direct Juniper-specific supervision while retaining a narrow platform scope.

Together, these studies justify testing compact specialized models while leaving attribution questions open. Comparisons against different base models change more than training alone,

while vendor transfer depends on what the adaptation data contain [7], [8], [44], [45]. Data provenance and partitioning are as important as aggregate dataset size when judging whether a reported improvement generalizes. Lira et al. provide particularly useful fine-tuning evidence because they test how an adapter trained on one vendor performs on a vendor it was not trained on.

### 2.4.3 Retrieval-Augmented Generation Using Vendor Documentation

Ouni et al. directly examine RAG as a way of supplying external knowledge at inference time, where the organization of that knowledge shapes what generation can use [9]. Hierarchical RAG (HRAG) preserves relationships among chapters, sections and content in vendor documentation. They compare ordinary retrieval, hierarchical retrieval, adaptation of the retrieval encoder and an additional reranking variant while retaining Llama 3.2 3B as the generator. The fine-tuned component is the e5-small-v2 retrieval encoder.

HRAG performs best with the adapted hierarchical retriever, while reranking adds no further gain [9]. The Cisco-focused evaluation uses expert and LLM assessment for 17 queries and LLM judging alone for 144 queries. The results support improved access to relevant documentation in the tested setting; device acceptance and behavioral correctness require separate validation. HRAG provides the cleanest retrieval evidence because it holds the generator fixed, although its larger evaluation uses LLM judging.

Example retrieval and manual retrieval also provide different kinds of guidance. Mekrache et al. retrieve similar, previously validated intent–output pairs from a knowledge base and place them into the prompt before a separate validation stage [3]. fastCEGS retrieves configurations whose intents and topology can inform a new synthesis task [36]. Such examples demonstrate existing combinations of decisions, whereas manuals provide syntax and constraints that the generator must assemble. Retrieval should be interpreted according to what kind of knowledge it supplies.

Retrieval can also expose configuration structure, not merely return prose. LCLE organizes manual-derived workflows and syntax into connected representations, allowing retrieval to preserve command dependencies and view hierarchy [39]. ConfigTransLE similarly demonstrates why vendor translation requires contextual organization: commands must be placed in valid target-side views and given the required prerequisites [15]. Its simulator evaluation shows that acceptance depends substantially on the surrounding execution

context. These studies make a common point through different tasks: retrieving an individually relevant command is insufficient if the generation process loses how that command relates to others.

Wireless intent translation adds a related but distinct example. The HIR-based LIT framework separates policy-sequence generation from parameter-value generation and uses manual-derived knowledge and hierarchical retrieval in those stages [46]. Its evaluation concerns policy and parameter outputs, alongside simulated resource-management objectives. This supports studying documentation organization and knowledge use, but its scores cannot serve as router CLI accuracy. It also differs from HRAG in its learning components, so their shared use of hierarchy does not make them the same intervention.

Across these results, retrieval failures should be separated from failures to apply retrieved information. NetConfEval's documentation-selection limitations are discussed in Section 2.3.2 [1]. LCLE and ConfigTransLE show why selection must also preserve structure [15], [39]. For Cisco and Juniper generation, document relevance, platform specificity and dependency coverage are intermediate questions; the final question remains whether the resulting configuration is correct in context.

### 2.4.4 Combined Fine-Tuning and Retrieval for Multi-Vendor Configuration

Fine-tuning and retrieval could be complementary because they supply knowledge through different channels. A model may learn recurring relationships between requirements and commands, while retrieval provides details specific to a platform or requested feature. ConfGen combines these mechanisms using a LoRA-adapted Llama2-13B generator, NetBERT-based retrieval, reranking and syntax-constrained decoding [10]. Its training uses 17,000 Huawei and Cisco intent–command pairs, while the documentation database covers Cisco, Huawei, Nokia and Juniper. The system provides multi-vendor evidence without giving every vendor equivalent generator adaptation.

That asymmetry is central to interpreting Cisco–Juniper relevance. ConfGen's Juniper coverage involves retrieved knowledge and transfer beyond the vendors represented in its stated fine-tuning pairs [10]. PreConfig instead includes Cisco and Juniper within a multi-task adaptation framework [7]. Neither comparison alone identifies whether a particular Juniper gain comes from training exposure, better retrieval or a more constrained output process.

Lira et al.'s cross-vendor results reinforce the need to examine training distribution and show that the number of supported vendors is not a sufficient generalization measure [8].

Component ablations help, but the question must match the ablation. ConfGen examines reranking and syntax-constrained decoding [10], while HRAG isolates retrieval variants with a fixed generator [9]. Outside networking, Ovadia et al. compare unsupervised fine-tuning with retrieval as knowledge-injection strategies and find that retrieval consistently performs better, for both previously seen and new knowledge [47]. Because their fine-tuning was unsupervised, the result does not settle the question for supervised, instruction-based adaptation on intent–configuration pairs. To our knowledge, that controlled question has not yet been answered for matched Cisco-and-Juniper configuration generation. Cross-paper contrasts support feasibility of combination more strongly than a general estimate of its incremental benefit.

Table 2.1. Vendor Coverage by Study

| Study | Vendors covered | How the vendors are covered |
|---|---|---|
| PreConfig [7] | Cisco, Juniper | Fine-tuning and evaluation using both vendors; includes generation, analysis, completion and cross-vendor translation. |
| SLM_netconfig [8] | Cisco, Huawei | Initially fine-tuned on Cisco; evaluated on Huawei before and after additional Huawei fine-tuning. |
| HRAG [9] | Cisco | Cisco documentation supports retrieval and retrieval-encoder fine-tuning; generation is evaluated on Cisco tasks. |
| ConfGen [10] | Cisco, Huawei, Juniper, Nokia | Generator fine-tuning uses Cisco/Huawei; retrieval and configuration-generation evaluation cover all four vendors. |
| ConfigTransLE [15] | Cisco, Juniper, Huawei, Nokia | Evaluates Cisco→Huawei, Juniper→Huawei and Huawei→Nokia configuration translation. |
| LLM-NetCFG [33] | Cisco | Configuration-generation and verification experiments use a four-router Cisco topology. |

| Study | Vendors covered | How the vendors are covered |
|---|---|---|
| fastCEGS [36] | Cisco | Cisco configuration examples support retrieval; synthesis experiments generate and verify Cisco router configurations. |
| ShowNet study [19] | Cisco, Juniper, Huawei, A10 Networks | Reports live diagnostic CLI use across these vendors; configuration modification is disabled. |
| LCLE [39] | Cisco, Huawei | Vendor manuals support structured retrieval and configuration-sketch evaluation; parameter completion is excluded. |

*Note. Vendor coverage does not imply equivalent training, testing or deployment evidence. Cross-vendor translation, transfer to another vendor and operational access to multiple vendors are distinct forms of evidence. Only studies with documented vendor-specific data, experiments or operational use are included.*

Complementarity remains an empirical question. Retrieval might supply missing vendor knowledge; repeat information already learned or introduce irrelevant context; adaptation might improve document use while also strengthening vendor bias. Cisco and Juniper results need separate reporting to show whether an aggregate gain is shared across vendors.

## 2.5 Validation and Iterative Correction

### 2.5.1 Syntactic and Logical Validation

Validation has meaning only in relation to the property checked. A JSON schema checks representation structure, a configuration parser checks supported syntax, and a behavioral test checks selected consequences of the resulting network state. LCLE's syntax validity ratio measures whether commands follow documentation-derived syntax graphs [39]. This is informative for sketch construction, but parameter completion and network behavior remain outside that metric. A high syntax score should retain its command-level meaning without becoming a claim of operational correctness. In this study, syntactic accuracy means that a configuration conforms to recognized target-vendor syntax, while logical accuracy means that

it satisfies the intended network requirement and, where applicable, the corresponding behavioral properties, such as reachability and isolation, when assessed using appropriate validation tools such as Batfish [48].

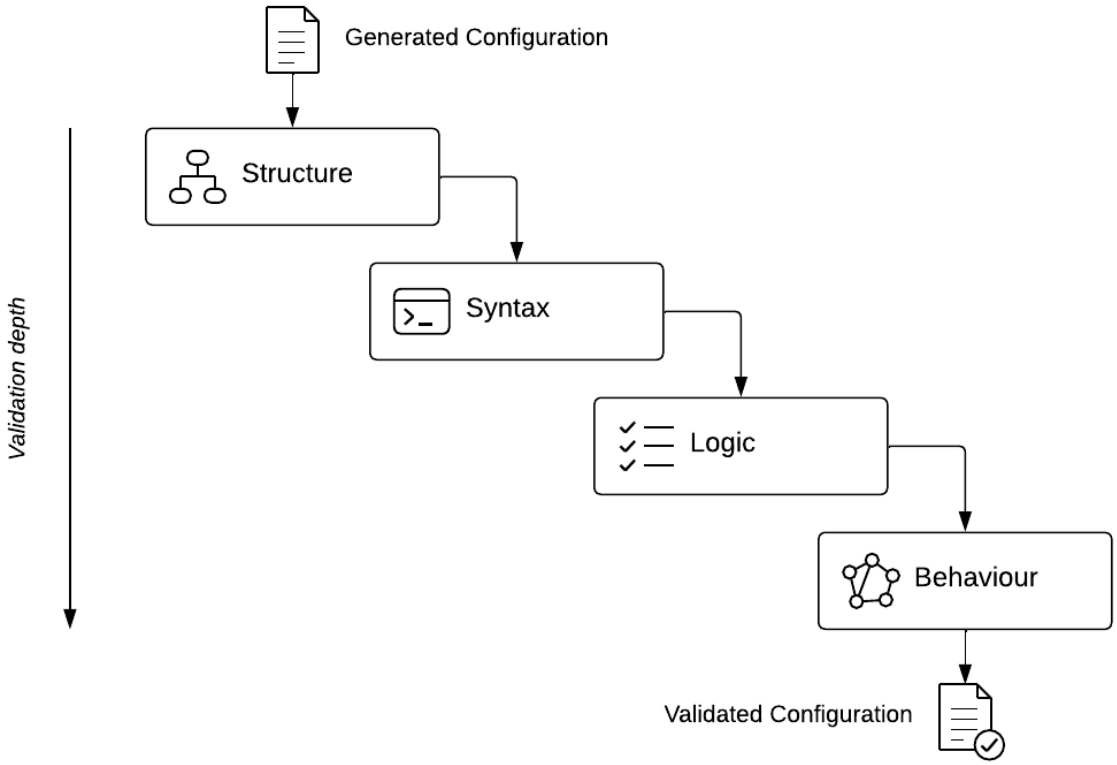



Figure 2.3. Validation Levels for LLM-Generated Network Configurations

ConfigTransLE offers a particularly clear demonstration of contextual acceptance. Its Juniper-to-Huawei simulator test evaluates 401 generated command-node occurrences across 114 fragments. Acceptance is 81.80% when commands receive valid scaffolding and prerequisite objects, 68.08% under arrangement-aware validation and 48.13% when submitted with only their generated ancestor context [15]. These are command-node acceptance rates, not percentages of fully successful fragments. The comparison shows that the execution context contributes materially to apparent correctness, even before end-to-end behavior is considered.

Model-based verification checks another layer. Batfish [48] analyzes configuration semantics and resulting network behavior from device configurations without relying on text similarity alone. ConfGen compares predicted and reference configuration changes from the same initial state using Batfish [10], while fastCEGS combines syntax, local topology and global

checks before final Batfish validation [36]. These methods detect policy and state violations that textual similarity can miss; their scope is defined by the properties encoded in the analysis and the semantics supported by the verifier. Verifier support also differs across vendors and features, and Junos configurations can be expressed either hierarchically or as set commands. For Cisco IOS and Juniper Junos comparisons, the verifier's coverage of the relevant constructs must be confirmed separately for each vendor.

Requirement selection is consequently part of validation design. LLM-NetCFG uses intent classification to guide the checks applied, and its reported misclassification cases illustrate how the pipeline can check syntax without the intended requirement-specific condition [33]. A configuration may pass an implemented check while the original request remains insufficiently assessed. This does not invalidate the checking tool; it identifies a dependency between interpretation and the properties submitted to it.

Mondal et al. provide the foundational LLM example of verifier-guided router correction. Their Verified Prompt Programming approach, proposed as the COSYNTH system, combines GPT-4 with Batfish [48], Campion [31], topology checks and localized natural-language feedback for Cisco-to-Juniper translation and a six-router BGP no-transit policy [6]. The experiments end with verified configurations and reduce manual prompting, but the evidence is preliminary: COSYNTH was not implemented as a fully automated system, API calls were simulated manually, only two use cases were tested, and some errors still required human intervention.

Finally, feedback and final assessment should be distinguished. LCLE uses its syntax machinery both to guide correction and to measure syntactic validity [39]. When the same tool guides correction and scores the result, an outcome the correction process does not see is needed to judge the benefit of that feedback.

### 2.5.2 Tool-Assisted and Agentic Correction

Agentic AI does not currently have a single universally accepted technical definition. In this study, agentic behavior is defined operationally as an LLM participating in a goal-directed feedback loop in which it receives observations from external validation tools, reasons over the returned feedback and selects revised configuration actions. The proposed workflow therefore represents bounded agentic behavior rather than unrestricted autonomous network operation.

LLM-NetCFG is best characterized as tool-assisted correction within fixed orchestration. The workflow generates configurations, obtains verifier feedback and requests regeneration until verification succeeds or a predefined limit is reached [33]. The model chooses revised content, while the orchestrator determines the broad sequence and stopping conditions.

The fine-tuned small-model system also places generation inside an orchestrated workflow with verification feedback and a bounded iteration process [8]. Zaid et al. go further by distributing network-management functions across multiple LLM agents, showing that multi-agent organization is a distinct architectural choice and is not a synonym for any correction loop [16]. Improvements in either case can arise from the generator, prompts, additional attempts or feedback, so first-pass and corrected outputs require matched attempt budgets for attribution.

Feedback granularity further affects what can be inferred from iterative correction. Huang et al. show that self-correction without reliable external feedback does not consistently improve LLM performance on general reasoning tasks [49], whereas Self-Refine demonstrates that iterative feedback and revision can improve outputs across several language-generation tasks [50]. These findings suggest that the effectiveness of correction depends not only on repeated generation but also on the quality and grounding of the feedback provided. In tool-assisted settings, externally derived feedback can provide more informative correction signals [51], with execution feedback serving a comparable role in code generation [52]. Consequently, evaluation should distinguish improvements attributable to feedback from those arising simply from additional sampling opportunities. This can be achieved by comparing feedback-guided correction against an unguided retry baseline under an equivalent generation budget. The pass@k metric used in code-generation evaluation provides a useful basis for this control by measuring whether at least one successful output is obtained within (k) sampled candidates [53]. For the proposed iterative validation pipeline, this principle can be extended by reporting cumulative success within a fixed correction budget alongside the number of correction attempts, validation-tool calls, computational cost, and any regressions introduced during successive revisions.

fastCEGS provides more differentiated feedback. Syntax, local and global checks target different failures, and the generation process receives information linked to detected errors before templates are completed through NetComplete [36], [29]. The architecture delegates

tasks according to their form: the LLM adapts examples, while formal machinery handles constrained completion and verification. However, the authors identify dependence on suitable examples and the capabilities of the underlying synthesizer. Iteration can repair a candidate within that supported space; it cannot be assumed to overcome missing coverage or an unrepresentable requirement.

Not every retrieval-assisted architecture implements a continuous correction loop. ConfGen explicitly identifies its current operation as open-loop pre-deployment generation and discusses runtime monitoring and feedback as future extensions [10]. ConfigTransLE includes an adviser that analyzes errors and supports an additional guided revision [15]. Its task begins from an existing configuration to translate, so command acceptance remains distinct from broader intent fulfillment.

Adaptive tool choice is clearer in the ShowNet assistant, which uses conversation context to select documentation, command-line and ticket-related tools [19]. Yet configuration modification was disabled, so this is operational assistance and is not evidence of autonomous deployment. Cornetto supplies a complementary warning about repair: its evaluation measures both corrected violations and regressions, showing why a proposed fix must be assessed beyond the initially reported fault [54]. A follow-up study on the Cornetto benchmark gives models context retrieval, step-by-step editing and verifier feedback, and reports that this agentic setup improves on single-pass repair [55]. Together, these studies support bounded, observable descriptions of agency. Useful comparisons should identify who selects the tool, who decides the revision, what terminates the process and whether new violations are checked. They should also count failed attempts and tool calls, because additional opportunities to correct output consume resources even when final quality improves.

## 2.6 Evaluation of Correctness and Efficiency

### 2.6.1 Benchmarks and Evaluation Metrics

NetConfEval separates specification translation, API interaction, routing-code generation and low-level configuration generation[1], illustrating how benchmarks evaluate different objects. NetLLMBench evaluates commands in a Kathará emulated network [56] containing one router, two switches and three hosts [14]. Cornetto evaluates repair of existing faulty

configurations across substantially larger networks [54]. These tasks evaluate different capabilities and should not be read as a single estimate of intent-to-configuration ability. Repair additionally supplies a faulty starting point and reported violations, whereas generation may require constructing the initial configuration from requirements.

Metric interpretation must preserve the evaluated unit. Exact Match and lexical overlap measure correspondence with a reference, while PreConfig's ConfigBLEU adds configuration keywords and syntax-tree information [7]. ConfigBLEU captures structural agreement more directly than ordinary Bilingual Evaluation Understudy (BLEU); forwarding behavior requires a separate validation layer. LCLE likewise separates syntax validity, recall and precision, exposing disagreement that a syntax-only score can miss [39].

Execution-based assessment also requires attention to denominators. NetConfEval tests supported real-protocol configurations with FRRouting [57] and computes normalized similarity for successfully loaded outputs; successful loading and similarity must be read together [1]. Its invented RIFT configuration language cannot be validated by a corresponding FRRouting implementation and is assessed differently [1]. Reporting only a similarity value would conceal these distinctions. NetLLMBench adds functional emulation, but its small task set limits claims about broader topology and policy complexity [14].

Cornetto makes non-regression part of repair evaluation. Its Fix Score divides fixed violations by the combined count of fixed, unfixed and regressed violations. In its best reported setting, the Fix Score is 57.8%, while 25.5% of cases are completely successful without regressions [54]. These percentages answer different questions: neither means that 57.8% of networks were fully repaired. Cornetto provides the most demanding correction evidence because its evaluation counts regressions as well as fixes.

Zheng et al. show that strong LLM judges can align well with human preferences while retaining biases and position effects [58], illustrating why an informative evaluation needs several distinct outcomes and judge-based scores require their own validation. This is relevant to HRAG, where the larger dataset relies on LLM judging [9]. Reference agreement, syntax or acceptance checks, and model-based property checks should remain separate outcomes; repeated runs and uncertainty estimates can further distinguish stable differences from generation variance.

### 2.6.2 Computational Cost and Latency

ConfGen reports retrieval, reranking and decoding timings on an RTX 3090 [10], whereas LLM-NetCFG reports a longer Central Processing Unit (CPU)-based workflow [33], showing why resource boundaries matter when interpreting quality improvements. The figures are not directly comparable because hardware, output length and timed stages differ, and generation-only timing omits later validation or repair.

LLNet shows that prompting choices affect both performance and operating expenses. More demonstrations increase the material processed by the model, while local and remote models have different accounting arrangements [37]. Its reported cost comparisons should be read within their stated measurement scope, without being converted into general deployment savings. A locally hosted model still requires computational resources even when it incurs no per-request API charge. Conversely, training expenditure is not a recurring charge on every request, although it matters when amortized over actual usage.

Pipeline comparisons should distinguish preparation from execution. Fine-tuning, document processing and index construction are preparatory activities; retrieval, generation, validation and correction recur during use. This distinction follows from the architectures of PreConfig, HRAG and the correction systems [7], [8], [33]. It suggests reporting both the cost of establishing a pipeline and the cost of obtaining a successful result. Counting only accepted outputs while ignoring failed attempts would make a retry-heavy system appear cheaper than its full workload warrants.

Baseline fairness also extends beyond model choice. NetLLMBench can call a master Llama3-70B model to repair malformed JSON, so assisted formatting must be considered when interpreting a tested model's outcome [14]. Cornetto distinguishes full-context, retrieval and oracle settings; an oracle given the affected files benefits from information that ordinary operational diagnosis does not necessarily have [54]. Such settings answer useful experimental questions, but should not be silently merged into an unassisted-model ranking.

The evidence supports conditional trade-offs and no single best pipeline. A defensible comparison holds task distribution, model version, demonstrations, decoding conditions and correction budget consistent where possible. Repeated runs, failed attempts, tool calls, end-to-end latency and cost per successful configuration then connect resource use to the quality actually obtained.

### 2.6.3 Evidence from Operational Network Environments

The ShowNet case study evaluates an assistant used by 105 engineers during Interop Tokyo 2025 and exposes contextual demands that controlled benchmarks simplify [19]. It combines GPT-4.1 with retrieved operational documents, command-line access and ticket information. Its device access spans multiple products and vendors. Such a setting exposes the need to reconcile documentation with live state and ongoing troubleshooting beyond isolated tasks whose relevant context has already been assembled.

The observational design also changes how improvement should be attributed. Engineers chose when to consult the assistant and provided ratings on a best-effort basis [19]. The rated subset may differ from unrated interactions, and usefulness can reflect the operator's ability to formulate questions and interpret answers. This does not diminish the value of the operational evidence, but it prevents treating a positive rating as an independently verified technical outcome. Human participation is part of the system being observed.

The reported outcomes require careful qualification. Of 4,007 exchanges, 389 received ratings and 265 of those were positive, giving 68.1% positive ratings among assessed responses [19]. This is not a success rate for all exchanges. Likewise, the reported 85.1% command success concerns syntactic execution of CLI commands. Configuration modification was disabled. The study provides evidence of useful but imperfect operational assistance, not autonomous intent fulfillment or successful production configuration deployment. Among the studies reviewed here, ShowNet provides the least controlled evidence but the closest view of LLM assistance in live network operations.

PRESTO contributes a different form of operational evidence: experience with expert-authored generation for provider services [20]. Compared with an LLM assistant, it places less open-ended interpretation in the runtime path and more responsibility in predefined service knowledge. Neither study provides a controlled experiment against the other. Their joint relevance is that operational value can arise through different allocations of human effort, and that deployment context must accompany any claim of practical reliability.

Operational observation adds heterogeneous devices, incomplete context and human judgement, but often weakens experimental control. Model-based verification offers more repeatable evidence for encoded configuration properties, while operational evidence

reveals factors that controlled studies simplify. Claims about autonomy should remain bound by the highest level of evidence actually reported.

Table 2.2. Pipeline Component Coverage

| Study | Fine-tuning | RAG | Multiple LLMs evaluated | Multi-vendor | Validation | Correction loop | Time / cost |
|---|---|---|---|---|---|---|---|
| NetConfEval [1] | ✓ | ✓ | ✓ | × | ✓ | ✓ | – |
| PreConfig [7] | ✓ | NR | – | ✓ | – | NR | NR |
| SLM_netconfig [8] | ✓ | NR | – | ✓ | ✓ | ✓ | ✓ |
| HRAG [9] | – | ✓ | × | × | – | NR | NR |
| ConfGen [10] | ✓ | ✓ | – | ✓ | ✓ | × | ✓ |
| NetLLMBench [14] | × | × | ✓ | × | ✓ | ✓ | ✓ |
| ConfigTransLE [15] | NR | ✓ | ✓ | ✓ | ✓ | – | ✓ |
| LLM-NetCFG [33] | NR | NR | × | × | ✓ | ✓ | ✓ |
| fastCEGS [36] | NR | ✓ | – | × | ✓ | ✓ | ✓ |
| ShowNet [19] | NR | ✓ | × | – | – | × | NR |
| LLNet [37] | × | × | ✓ | × | ✓ | – | ✓ |
| LCLE [39] | NR | ✓ | ✓ | ✓ | ✓ | ✓ | – |
| HIR/LIT [46] | ✓ | ✓ | – | NR | – | NR | NR |
| Cornetto [54] | × | – | ✓ | NR | ✓ | – | ✓ |
| Agentic repair [55] | × | – | ✓ | NR | ✓ | ✓ | ✓ |

| Study | Fine-tuning | RAG | Multiple LLMs evaluated | Multi-vendor | Validation | Correction loop | Time / cost |
|---|---|---|---|---|---|---|---|
| **Proposed Study (planned)** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

*Note. ✓= major implemented/evaluated component; – = limited/supporting role; × = not used in the tested workflow; NR = not reported or insufficiently established. The proposed row denotes planned coverage across stages, not completed findings.*

Table 2.3. Key Studies, Remaining Gaps and Research Response.

| Study | Key contribution | Key limitation / remaining gap | How this research addresses the gap |
|---|---|---|---|
| PreConfig [7] | LoRA fine-tuning for Cisco and Juniper configuration tasks. | FT, RAG and validation are not isolated under matched conditions. | Controlled FT/RAG ablations across models and both vendors. |
| HRAG [9] | Hierarchical RAG improves retrieval from networking documentation. | Cisco-focused; limited technical validation of generated configurations. | Cisco/Juniper RAG evaluated for syntax, logic and cost. |
| ConfGen [10] | Combines LoRA, retrieval and constrained decoding for multi-vendor generation. | Training and retrieval vendor coverage differ; component effects remain intertwined. | Matched FT/RAG conditions with vendor-separated reporting. |
| COSYNTH [6] | Uses verifier feedback to iteratively correct router configurations. | Small, partly manual evaluation; feedback benefit and added cost are not isolated. | Bounded automated correction with matched retry and cost controls. |
| NetConfEval [1] | Benchmarks LLM network-configuration tasks using reference and executable evaluation. | Does not isolate FT, RAG and agentic correction under one matched design. | Common multi-vendor evaluation across all pipeline stages. |

## 2.7 Research Gap

Different studies have investigated fine-tuning, retrieval-augmented generation (RAG), structured generation, validation and iterative correction for LLM-based network configuration [6], [7], [8], [9], [10], [33], [36]. However, these components are generally evaluated using different models, datasets, vendors, prompts and evaluation methods, making their individual and combined effects difficult to isolate. Among the studies reviewed, no study was identified that evaluates fine-tuning, RAG and verifier-guided iterative correction together under a controlled, matched pipeline across Cisco IOS and Juniper Junos. Similarly, although several studies compare multiple LLMs [14], [54], limited evidence exists on how different model families respond to the same domain-adaptation and retrieval conditions. Controlled comparison of models such as Qwen and Llama under matched fine-tuning, RAG and evaluation settings is therefore needed to determine whether the effectiveness of these techniques is model-dependent.

A related gap concerns vendor coverage. Cisco appears across several fine-tuning, retrieval and verification studies, whereas Juniper Junos is represented less consistently, either through narrow single-platform datasets [45] or broader multi-vendor evaluations [7], [10], [15]. Consequently, limited evidence shows whether the same adaptation, retrieval and validation techniques behave consistently across Cisco IOS and Juniper Junos, where syntax, command hierarchy and configuration conventions differ.

Finally, verifier-guided correction has shown promise in Mondal et al., LLM-NetCFG and fastCEGS [6], [33], [36], but its independent contribution remains unclear because validation tools, retry strategies and experimental conditions differ. Improvements may arise from validation feedback itself or simply from additional generation attempts, while the associated computational overhead is rarely isolated. These gaps motivate a controlled investigation of fine-tuning, RAG and agentic validation with iterative correction across multiple model families and both target vendors, with accuracy and computational cost evaluated under matched conditions.

# Chapter 3: Aim and Objectives of Research

## 3.1 Problem Statement

Network configuration is a fundamental network-management function, but translating operational requirements into correct device-level configurations remains difficult because configurations are precise, vendor-specific and sensitive to small errors. Large Language Models (LLMs) allow these requirements to be expressed in natural language, but plausible output is not necessarily correct. Generated configurations may contain incorrect parameters, unsupported syntax, missing dependencies or logic that fails to achieve the intended network behavior. Mondal et al. identified syntax, topology and semantic errors in GPT-4 router configurations that required verifier feedback and, in some cases, human intervention [6]. Such errors make unverified LLM-generated configurations unsuitable for dependable deployment.

Three approaches are particularly relevant to this problem: domain-specific fine-tuning, which adapts model parameters using network-specific data [7]; Retrieval-Augmented Generation (RAG), which supplies external vendor knowledge during inference [9]; and agentic validation with iterative correction, in which validation feedback guides subsequent revisions [6]. Their effectiveness, however, is not consistent across settings. Supplying retrieved documentation did not improve every configuration scenario in NetConfEval [1], while a model adapted to Cisco transferred poorly to Huawei until additional vendor-specific fine-tuning was applied [8].

As reviewed in Chapter 2, existing studies use different models, datasets, vendors, prompts and validation procedures, making the individual and combined effects of these techniques difficult to isolate. It therefore remains unclear whether observed improvements arise from fine-tuning, retrieval, the underlying model family, additional generation attempts, validation feedback, or interactions among these factors. Evidence is also limited on whether the same techniques produce comparable gains across Cisco IOS and Juniper Junos. Computational overhead, particularly for iterative correction, is also rarely considered together with configuration accuracy.

This lack of controlled evidence creates a practical decision problem: researchers and network-engineering teams cannot determine whether the additional training, retrieval or

validation cost of a pipeline component produces a worthwhile improvement for a particular model and vendor. Open-weight models are particularly relevant because they support direct adaptation, controlled experimentation and local deployment without complete dependence on proprietary model services. The research problem addressed by this thesis is therefore the lack of controlled evidence on how domain-specific fine-tuning, RAG and agentic validation with iterative correction, individually and in combination, affect the syntactic accuracy, logical accuracy and computational cost of open-weight LLMs for Cisco IOS and Juniper Junos configuration generation.

## 3.2 Research Aim

The aim of this research is to systematically investigate the individual and combined effects of domain-specific fine-tuning, Retrieval-Augmented Generation (RAG), and an agentic AI workflow incorporating tool-assisted validation and iterative correction on the syntactic and logical accuracy of natural-language-driven Cisco IOS and Juniper Junos configuration generation using selected open-weight LLMs. Through controlled experimental comparisons, the study aims to quantify the contribution and computational cost of each technique and their combinations, and to use the resulting empirical evidence to design an effective LLM-based configuration-generation pipeline that balances configuration correctness with computational efficiency across the targeted vendors.

## 3.3 Research Questions

Accordingly, this thesis is guided by one overarching Main Research Question (MRQ), supported by two interrelated Sub-Research Questions (SRQ1 and SRQ2) that examine the principal components of the proposed investigation:

### 3.3.1 Main Research Question (MRQ)

How can an open-weight LLM-based pipeline be designed to translate natural-language network intents into syntactically and logically accurate Cisco IOS and Juniper Junos configurations while remaining computationally efficient?

### 3.3.2 Sub-Research Question 1 (SRQ1)

How do domain-specific fine-tuning and retrieval-augmented generation, individually and in combination, affect the syntactic accuracy, logical accuracy and computational cost of selected open-weight LLMs for Cisco IOS and Juniper Junos configuration generation?

### 3.3.3 Sub-Research Question 2 (SRQ2)

To what extent does an agentic AI workflow incorporating tool-assisted validation and iterative correction improve the syntactic and logical accuracy of LLM-generated Cisco IOS and Juniper Junos configurations, and what additional computational cost does it introduce?

## 3.4 Research Objectives

RO1: To develop a curated multi-source dataset for fine-tuning selected open-weight LLMs for Cisco IOS and Juniper Junos configuration generation.

RO2: To quantify the individual and combined effects of domain-specific fine-tuning and vendor-specific Retrieval-Augmented Generation (RAG) on the syntactic accuracy, logical accuracy and computational cost of selected open-weight LLMs.

RO3: To quantify the improvements in syntactic and logical accuracy achieved through an agentic AI workflow incorporating tool-assisted validation and iterative correction, and the additional computational cost introduced by the correction process.

RO4: To synthesize the experimental findings into an evidence-based open-weight LLM pipeline that balances syntactic accuracy, logical accuracy and computational efficiency for Cisco IOS and Juniper Junos configuration generation**.**

## 3.5 Alignment of Research Questions and Objectives

The research objectives are directly aligned with the research questions.

Objective 1 establishes the data foundation required for the domain-adaptation experiments, while Objective 2 directly addresses Sub-Research Question 1 by evaluating the effects of fine-tuning and RAG.

Objective 3 addresses Sub-Research Question 2 through the evaluation of the agentic AI workflow incorporating tool-assisted validation and iterative correction.

Objective 4 synthesizes these experimental findings to address the Main Research Question through the design of an evidence-based, computationally efficient LLM pipeline.

Table 3.1. Research Question and Objective Alignment

| Research Question | Research Objective(s) |
|---|---|
| MRQ | RO1–RO4 |
| SRQ1 | RO1, RO2 |
| SRQ2 | RO3 |
| MRQ synthesis/design | RO4 |

### 3.6 Research Originality and Expected Contributions

The expected originality of this research lies primarily in the controlled investigation and integration of techniques that have largely been examined separately in existing LLM-based network-configuration research. The study will quantify the individual and combined effects of domain-specific fine-tuning, vendor-specific Retrieval-Augmented Generation (RAG), and an agentic AI workflow incorporating tool-assisted validation and iterative correction under a consistent experimental framework. This design is intended to provide clearer evidence of the contribution of each component that can be obtained through comparisons across studies using different models, datasets, vendors and evaluation procedures.

The research is also expected to contribute multi-vendor evidence by examining these effects across both Cisco IOS and Juniper Junos and identifying whether improvements and trade-offs are consistent across the two environments. These experimental findings will subsequently be synthesized into an evidence-based open-weight LLM pipeline whose model adaptation, retrieval and validation components are selected according to their demonstrated effects on syntactic accuracy, logical accuracy and computational efficiency.

# Chapter 4: Research Design and Proposed Method

This chapter presents the research methodology and experimental design used to address the research questions and identified research gaps.

## 4.1 Research Approach

This study adopts a pragmatic quantitative experimental research design. The research focuses on measuring how different Large Language Model (LLM) architectures, model sizes, fine-tuning settings, Retrieval-Augmented Generation (RAG), validation methods and agentic workflows affect the accuracy and reliability of natural-language-to-network-configuration generation.

The study does not use questionnaires, interviews or surveys because it does not investigate human opinions or behavior. Instead, numerical data will be collected from controlled AI experiments. These data will include configuration accuracy, syntactic accuracy, logical accuracy, validation pass rate, training loss, latency, Graphics Processing Unit (GPU) usage, token usage, cost and correction success rate.

The aim is to determine, through measurable experimental comparison, which combination of dataset design, open-weight LLM, fine-tuning strategy, retrieval method and validation workflow provides the best performance for converting natural-language network intent into network configuration.

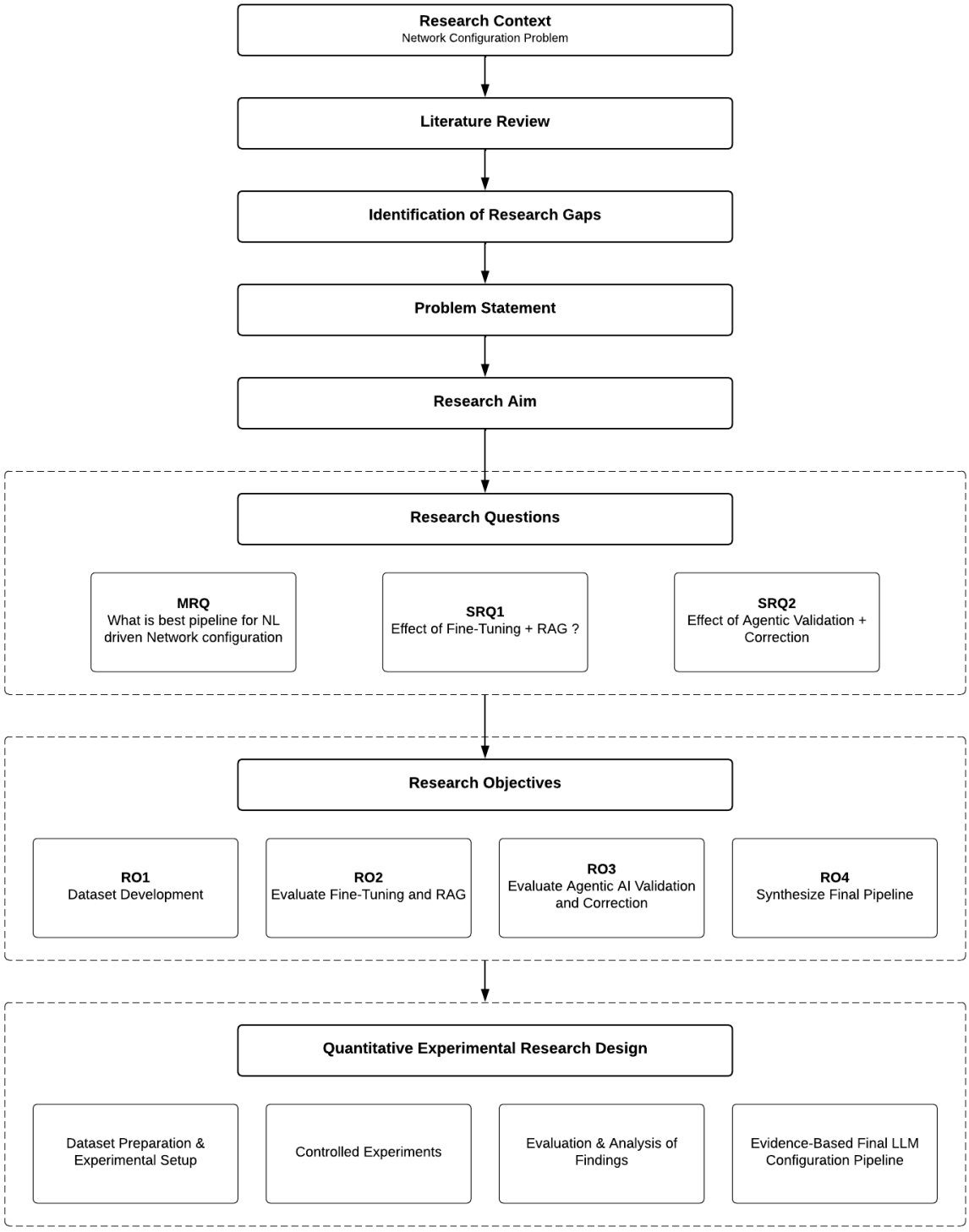



Figure 4.1. Research Methodology Framework

## 4.2 Dataset Development and Preparation

### 4.2.1 Data Sources and Acquisition

This study develops a multi-source, domain-specific dataset by combining existing networking datasets into a unified resource for supervised fine-tuning. Three publicly available datasets, associated with studies examined in the literature review, were selected for the initial experiments.

The Cisco IOS dataset [8], [59] provided 5,096 imported requirement–configuration records, of which 4,624 were retained after preparation. The Juniper NIT dataset [45], [60] contained 1,000 original question–context–answer records. Controlled augmentation expanded this collection to 4,091 records, with 4,044 retained after preparation. NetConfEval [1], [61] provided 3,358 records across four task subsets, of which 3,307 were retained.

The resulting initial dataset contains 11,975 examples. Additional sources may be incorporated to improve coverage, subject to quality assessment, relevance and applicable reuse permissions.

Table 4.1. Dataset Sources

| **Dataset** | **Vendor / Task** | **Retained Records** |
|---|---|---|
| Cisco IOS dataset [8], [59] | Cisco IOS configuration | 4,624 |
| Juniper NIT [45], [60] | Juniper Junos configuration | 4,044 |
| NetConfEval [1], [61] | Multi-task networking | 3,307 |
| Combined dataset | Multi-vendor / multi-task | 11,975 |

### 4.2.2 Dataset Content and Task Coverage

#### *4.2.2.1 Cisco IOS Dataset*

The Cisco dataset pairs natural-language configuration requirements with corresponding Cisco IOS commands [8], [59]. Its purpose in this study is to train models to generate device configurations from user requirements.

#### *4.2.2.2 Juniper Junos NIT Dataset*

NIT contains a natural-language question, a generic command context and a corresponding Junos answer. The original dataset targets the Juniper EX3300 platform [45], [60]. Its command contexts provide templates for controlled augmentation, while its question–answer pairs support supervised training.

#### *4.2.2.3 NetConfEval Dataset*

NetConfEval provides examples involving intent-to-JSON translation, requirement-conflict detection, routing-code generation and low-level configuration generation [1], [61]. The low-level configurations include FRRouting-based scenarios and are not equivalent to Cisco IOS or Junos commands. Its routing-code-generation subset supplies executable tests rather than reference code outputs and is therefore excluded from direct supervised fine-tuning.

The combined dataset consequently supports multiple networking tasks, rather than exclusively vendor-specific CLI generation.

### 4.2.3 Source-Specific Cleaning and Quality Control

Cisco records were processed to extract configuration commands and remove explanations, Markdown formatting, device prompts and repeated blocks. Empty records, operational-only responses and duplicates were excluded. Juniper records were screened for missing fields, defective examples and duplication. Suspect source records were not used as augmentation seeds.

NetConfEval subsets were assessed individually according to their input and reference-output structures. Records unsuitable for supervised training were excluded rather than assigned fabricated answers. Exclusions and their reasons were recorded.

These procedures improve structural consistency but do not establish whether a configuration executes successfully or achieves its intended network behavior.

### 4.2.4 Targeted Juniper Data Augmentation

Controlled augmentation was used to address the smaller size of the original Juniper dataset. A deterministic Python procedure used existing command contexts as templates and generated new examples by substituting suitable interface names, addresses, Virtual Local Area Network (VLAN) identifiers and other parameters.

Question wording was drawn from compatible examples sharing the same command template. Ambiguous or defective records were excluded from the augmentation seed pool. Generated examples were checked for template consistency, parameter alignment and duplication. Their source-record identifiers and augmentation details were retained for traceability.

These checks establish consistency with the source templates, not operational correctness. Any future AI-assisted augmentation will require separate verification before the generated examples are included.

### 4.2.5 Dataset Integration and Training Format

Prepared records were combined into a unified dataset containing a record identifier, source dataset, task, vendor, output type, input, reference output and parent group. The dataset was exported in Comma-Separated Values (CSV) and JSON Lines (JSONL) formats.

For supervised fine-tuning, each record was converted into a conversation containing a system instruction, a user message labeled TASK, VENDOR, OUTPUT_TYPE and INPUT, and an assistant message containing the reference output. Source identifiers and augmentation provenance were retained as metadata rather than included in the model prompt.

### 4.2.6 Partitioning and Leakage Checks

The prepared dataset was divided reproducibly into approximately 80% training, 10% validation and 10% internal testing. Juniper examples sharing a command-template group were assigned to the same partition. Cisco records were grouped using normalized questions, while NetConfEval records were split within their respective task subsets. Cross-partition checks identified no exact input–output duplicates, and a similarity audit flagged potentially overlapping examples for further review. The internal test set was reserved for final evaluation and excluded from training and model selection.

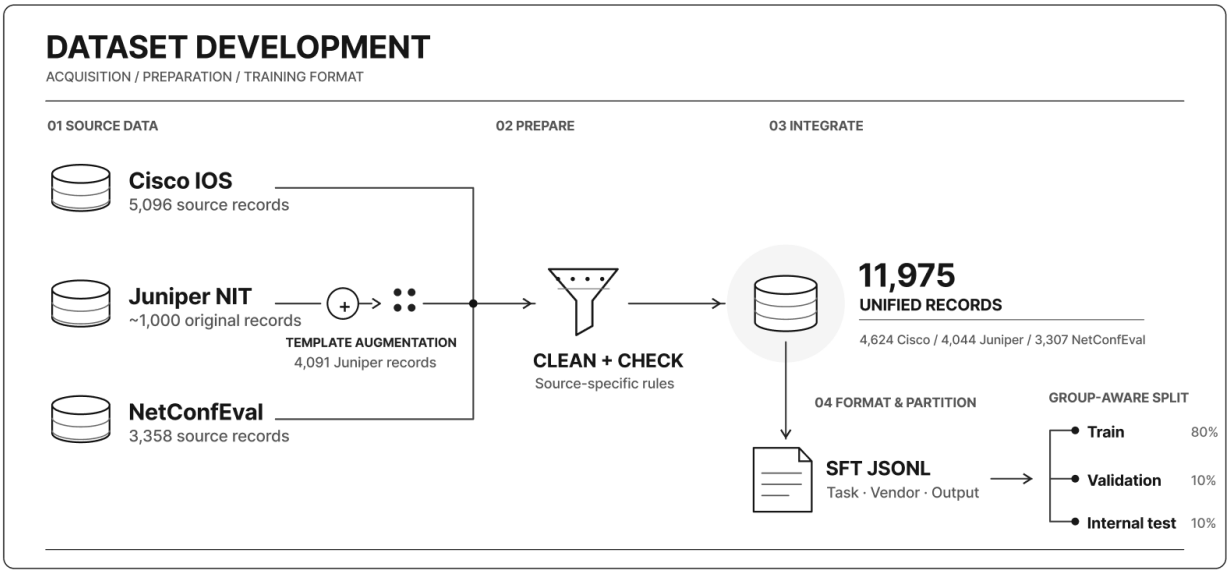



Figure 4.2. Dataset Development

## 4.3 Model Selection and Baseline Establishment

This study will investigate open-weight models from the Qwen and Llama families. Qwen is selected for its documented code-generation and reasoning capabilities [62]. Llama is selected for its instruction-following, coding and reasoning capabilities [63]. Multiple model sizes will be considered according to available computational resources. Before fine-tuning, each candidate will undergo baseline evaluation using consistent prompts, generation settings and validation examples. Following fine-tuning, model performance will be compared using the same evaluation procedures. One model from each family will be selected based on validation results for subsequent RAG experiments. The reserved test set will remain excluded from model selection.

## 4.4 Model Fine-Tuning and Optimization

### 4.4.1 Model Adaptation Procedure

The selected Qwen and Llama candidates will undergo supervised fine-tuning using the prepared multi-vendor dataset. Fine-tuning adapts a pretrained model to a specific task by training it on examples containing an input and its expected output. In this study, natural-language network requirements will serve as inputs, while their corresponding configurations or structured representations will provide training targets. Parameter-efficient adaptation will be employed to modify a limited set of trainable parameters while retaining the pretrained model's underlying knowledge [41], [42].

### 4.4.2 Controlled Fine-Tuning Experiments

Separate experiments will be conducted for each candidate model to examine how training conditions influence performance. Selected hyperparameters, including learning rate, adaptation rank, batch size and training epochs, will be varied systematically within available computational constraints. The same dataset partitions and evaluation procedures will be maintained to support comparisons across experimental conditions. Training and validation losses will be recorded throughout each run to examine convergence and identify potential overfitting. Each fine-tuned model will also be compared against its corresponding pretrained baseline to establish the improvement attributable to domain-specific adaptation.

### 4.4.3 Validation-Guided Optimization

Optimization will use validation results rather than the reserved test set. Training settings and checkpoints will be selected according to predefined performance criteria, with attention to both configuration quality and computational efficiency. The strongest validated fine-tuned model from each family will subsequently advance to the RAG experiments. The test partition will remain excluded from optimization and model selection to preserve its role in final evaluation.

### 4.5 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) will be investigated as a method for supplying external, vendor-specific knowledge to the selected fine-tuned models during configuration generation. Unlike fine-tuning, RAG incorporates retrieved information at inference time without modifying model parameters [43].

A knowledge base will be developed from publicly accessible Cisco and Juniper documentation, subject to applicable permissions and usage conditions. Relevant command references, configuration examples and technical explanations will be collected and cleaned. Documents will then be divided into meaningful passages, associated with source information and converted into embeddings for retrieval. Command hierarchies and relevant prerequisites will be preserved during this process.

For each network requirement, the retrieval system will identify relevant passages from the appropriate vendor collection and provide them alongside the original intent. The selected Qwen and Llama models will be evaluated both with and without RAG under otherwise consistent experimental conditions.

Retrieval settings will be refined using validation data. Comparisons will examine whether retrieved knowledge improves configuration correctness and reduces missing or incorrect commands, while also measuring latency and inference cost. Retrieved sources will be recorded to support error analysis and reproducibility.

### 4.6 Agentic Validation and Feedback

An agentic workflow will be investigated to determine whether automated validation and iterative correction improve configuration reliability. Previous research has explored LLM-based network configuration systems that integrate generation and verification, providing a basis for investigating this approach [6], [33].

The experiment will compare single-step configuration generation against a workflow incorporating generation, validation, feedback and correction. The selected models will initially generate configurations from network requirements, with and without RAG.

Batfish will be used to examine configuration parsing and supported network properties within appropriate device and topology context [48]. Parsing outcomes, warnings and identified network problems will provide feedback for correction. However, successful parsing will not be treated as proof of complete syntax validity or correct network behavior. Unsupported features and inconclusive results will be recorded separately. Formal network verification provides valuable evidence but has limitations that require complementary testing [64]. When validation identifies an actionable problem, the generator will receive the relevant feedback and attempt to revise its output. This process will continue until the configuration satisfies the specified checks or reaches a predefined correction limit.

An LLM-based evaluator may also be explored as a supplementary method for assessing alignment with the original intent. Its assessments would require comparison against independently verified examples before inclusion in the primary evaluation.

The experiment will measure first-pass success, final success, correction attempts, repair success, latency and additional computational cost. Comparisons will establish whether observed improvements justify the resources required by iterative correction.

### 4.7 Benchmark Development and Evaluation Framework

An independent multi-vendor benchmark will be developed to provide a consistent evaluation basis across the experimental stages of this study. It will contain at least 200 test cases, with a minimum of 100 cases each for Cisco IOS and Juniper Junos, covering representative tasks such as interface configuration, IP addressing, VLANs, routing and access control. Each case will define a natural-language intent, expected configuration requirements and predefined evaluation criteria. Benchmark cases will remain separate from training and optimization data, and their construction will be documented to support reproducibility and minimize ambiguity or data leakage.

The purpose of this benchmark is not to replace existing evaluation resources, but to enable controlled comparison of fine-tuning, RAG, validation and correction under the same task distribution and evaluation conditions. Existing benchmarks such as NetConfEval [1] and NetLLMBench [14] will also be used where their tasks and evaluation settings are applicable. Because these benchmarks differ in scope, topology, task definition and evaluation method,

their numerical results will be analyzed independently rather than treated as directly comparable with the proposed benchmark. This combined approach provides both controlled internal comparison and external reference evidence.

### 4.8 Quantitative Analysis and Pipeline Selection

The findings from the preceding experiments will be synthesized to design and propose a final pipeline for natural-language-driven network configuration using LLMs and agentic AI. The pipeline architecture will be determined by the measured contributions of fine-tuning, RAG and iterative validation, rather than assuming that every component improves performance.

Quantitative analysis will compare pretrained and fine-tuned models, generation with and without RAG, and single-step generation against agentic correction. Performance will be measured using configuration correctness, task completion, validation success, correction success, latency and computational cost.

Results will be analyzed separately for Cisco and Juniper to identify vendor-specific differences. Configuration failures will be categorized and their frequencies calculated to reveal recurring limitations. Where appropriate, confidence intervals and statistical tests will assess observed performance differences.

Pipeline components and operating settings will be selected using predefined criteria and validation results. The resulting architecture will then be assessed using the reserved test set and available functional testing evidence. The final proposal will specify the selected model, retrieval approach and validation workflow, alongside their demonstrated performance, resource requirements and limitations.

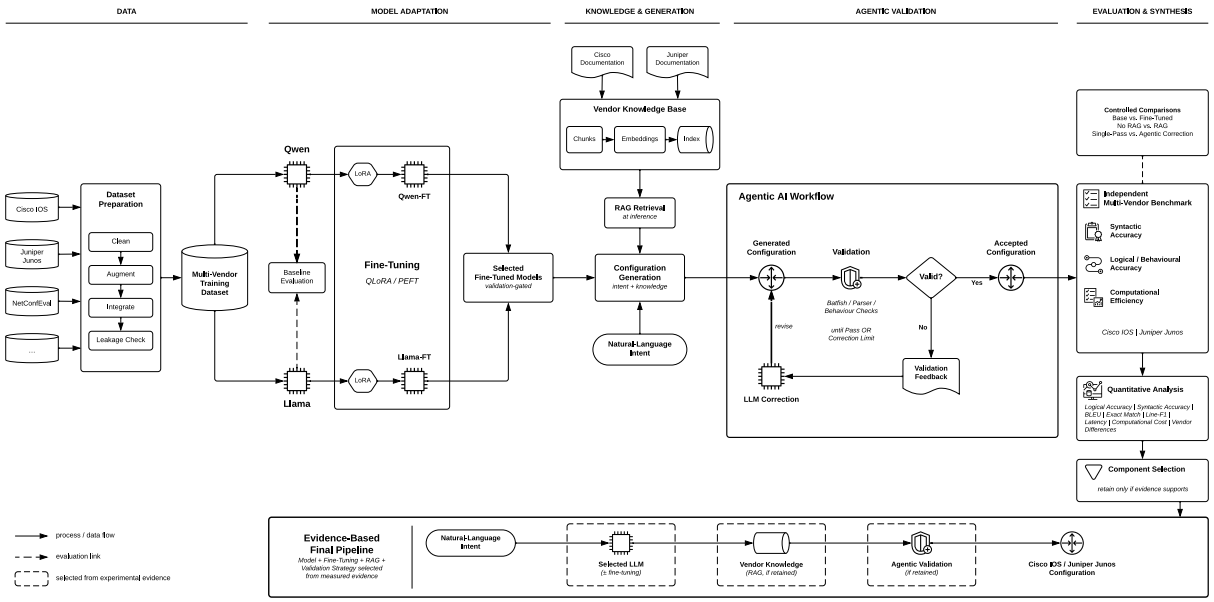



Figure 4.3. Proposed Experimental Framework for LLM-Based Network Configuration

## 4.9 Evaluation Metrics

This section defines the reference-based, validity, and computational measures used across the experiments.

### 4.9.1 Reference-Based Metrics

**Exact Match (EM).**

Measures the proportion of generated configurations that exactly match the corresponding benchmark reference. [10], [65]

$$EM = \frac{N_{\text{exact}}}{N}$$

*N_exact = number of generated configurations that exactly match the benchmark reference; N = total number of test cases.*

**Bilingual Evaluation Understudy (BLEU).**

Measures n-gram overlap between generated and reference configurations.

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

*BP = brevity penalty; $p_n$ = modified n-gram precision; $w_n$ = weight assigned to each n-gram order.*

**Line F1.**

Measures command-line overlap between the generated and reference configurations, while penalizing missing and unnecessary lines. [7], [10], [39] (line-level adaptation)

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

*TP = correctly generated reference lines; FP = unnecessary/incorrect generated lines; FN = required reference lines that are missing.*

### 4.9.2 Configuration Validity Metrics

**Syntax Accuracy.**

Assesses whether the generated configuration commands conform to recognized configuration syntax. [39]

$$\text{Syntax Accuracy} = \begin{cases} 1, & \text{if all generated commands are valid} \\ 0, & \text{if at least one command is incomplete or unusual} \\ -1, & \text{if at least one command is invalid} \end{cases}$$

*The score distinguishes fully valid, partially acceptable, and invalid configurations.*

**Logical Accuracy.**

Assesses whether the generated configuration satisfies the intended network requirement. [39]

$$\text{Logical Accuracy} = \begin{cases} 1, & \text{if configuration } C \text{ fully satisfies requirement } R \\ 0, & \text{if configuration } C \text{ partially satisfies requirement } R \\ -1, & \text{if configuration } C \text{ does not satisfy requirement } R \end{cases}$$

*C = generated configuration; R = original network requirement.*

### 4.9.3 Computational and Cost Metrics

Inference latency, end-to-end execution time, GPU/CPU utilization, memory use, correction iterations, and monetary cost will be recorded to quantify the computational overhead of each pipeline [8], [10], [33], [37].

## Chapter 5: Research Requirements Analysis and Specification

This chapter specifies the models, software, compute resources, validation tools, and reproducibility requirements for the study.

### 5.1 Language Models

#### 5.1.1 Initial Experimental Models

The initial experiments use Qwen2.5-3B-Instruct [66] and Llama-3.2-3B-Instruct [67] as the primary open-weight models. Their similar parameter scale makes repeated fine-tuning practical within the available computing budget and provides a controlled starting point for comparing two model families without making model size the main experimental difference.

#### 5.1.2 Extended Model Candidates

The framework is not restricted to these models. Later experiments may consider stronger, later-generation Qwen and Llama models when they can be fine-tuned within available GPU capacity and financial limits. Selection will consider model size, licensing, compatibility with the established training stack and the cost of repeating the full evaluation protocol. Larger models will only be introduced where a meaningful comparison with the initial experiments remains feasible.

### 5.2 Dataset Preparation and Augmentation Tools

Dataset engineering is implemented through custom Python scripts. The current preparation code uses Pandas, JSON utilities, regular expressions, pathlib, Requests, the Hugging Face Hub interface, NumPy and scikit-learn. These tools support cleaning, normalization, task and vendor labeling, deterministic partitioning, duplicate detection, similarity-based leakage checks and CSV or JSONL export. Juniper augmentation is also performed through existing custom Python scripts using controlled template-based generation while retaining source

relationships and provenance. This section therefore documents tools already present in the implemented data pipeline rather than proposed alternatives.

## 5.3 Fine-Tuning Tools and Libraries

This section summarizes the software libraries and parameter-efficient methods used for model adaptation.

### 5.3.1 PyTorch and Hugging Face Framework

The training workflow is implemented with PyTorch and the Hugging Face ecosystem. PyTorch provides tensor computation and Compute Unified Device Architecture (CUDA)-based GPU execution. Transformers load the Qwen and Llama models and tokenizers and applies their native chat templates. The implemented environment also includes Datasets, Transformer Reinforcement Learning (TRL), Parameter-Efficient Fine-Tuning (PEFT), bitsandbytes, Accelerate and Hugging Face Hub. Together, these libraries support supervised fine-tuning, dataset handling, adapter training, quantized model loading and model access.

### 5.3.2 Parameter-Efficient Fine-Tuning

The study uses PEFT rather than updating every base-model parameter. LoRA keeps the pretrained weights frozen and learns smaller low-rank updates. Its original study reported performance comparable to, and in some tested cases better than, full fine-tuning while substantially reducing trainable parameters and GPU memory demand [41]. This motivates LoRA as an efficiency method but does not imply equal performance for every network-configuration task. QLoRA extends the approach by retaining the base model in low-bit form while training LoRA adapters. Dettmers et al. demonstrated fine-tuning of a 65-billion-parameter model on a single 48 GB GPU while preserving full 16-bit fine-tuning task performance in their evaluated setting [42]. The present implementation follows this approach through PEFT adapters and bitsandbytes 4-bit quantization.

### 5.3.3 Training Outputs and Model Storage

Transformer Reinforcement Learning (TRL) provides the supervised fine-tuning trainer used by the current code, while Accelerate remains part of the execution stack. Adapters, tokenizers and experiment outputs are stored separately so completed runs can be reloaded.

The Hugging Face Hub will be used to store trained adapters and, where a full export is produced, the resulting fine-tuned model artifacts.

### 5.4 Computing and Development Environment

The Cisco-only experiments used an NVIDIA A100-SXM4-80GB on RunPod, while the multi-task experiments used an NVIDIA L40 on Thunder Compute. GPU model, memory, runtime and experimental cost are recorded. No universal parameter-count-to-GPU rule is adopted because memory demand also depends on quantization, sequence length, batch size and training method. The QLoRA 65B-on-48 GB result is therefore treated as evidence from one specific setup, not as a general sizing rule [42]. Development uses Python, Linux, Jupyter Notebook and Visual Studio Code with remote execution where required.

### 5.5 Retrieval-Augmented Generation Tools

RAG remains a proposed extension and has not yet been integrated into the experiments. The first implementation will therefore be deliberately simple. Permitted Cisco and Juniper technical material will be prepared as retrievable passages. A network requirement will be used to retrieve relevant vendor material, which will then be supplied with the original requirement to the model. This follows the general RAG principle of combining generation with retrieved external information [68]. Sentence Transformers and Facebook AI Similarity Search (FAISS) are candidate tools because Sentence-BERT supports sentence embeddings for similarity comparison [69], while FAISS supports vector similarity search [70]. Their use will remain provisional until a small pilot confirms suitability.

### 5.6 Automation and Agentic Workflow Tools

The agentic component is also planned rather than fully implemented. The initial design will use explicit Python orchestration so that each stage can be measured. A generated configuration will be validated, actionable feedback will be returned to the model, and a revised output may be produced until the checks pass or a correction limit is reached. Related network-configuration research has explored automated generation and verification loops using LLMs [33]. No external agent framework or device-automation library is fixed at this stage.

## 5.7 Network Verification and Simulation Tools

This section describes the tools considered for configuration analysis and experimental network validation.

### 5.7.1 Batfish and PyBatfish

Batfish is the principal proposed tool for model-based configuration analysis where the relevant syntax is supported. It analyzes network configurations and forwarding behavior, with research work demonstrating its use for scalable network analysis [71]. PyBatfish provides the Python interface for integrating these checks into experimental scripts. Preliminary development launched Batfish successfully, although an early parsing problem occurred. Later use will therefore require complete network snapshots and checks of supported syntax rather than assuming every generated command can be analyzed.

### 5.7.2 GNS3 as a Possible Complement

GNS3 remains a possible complementary environment for virtual-device testing. It will only be incorporated if suitable Cisco and Juniper images, licensing conditions and a stable experimental workflow can be established. It is therefore treated as optional rather than a guaranteed evaluation stage [72].

## 5.8 Evaluation and Analysis Tools

The existing evaluation notebooks use custom Python code for Exact Match, Normalized Exact Match, line-level precision, recall and F1, BLEU, Recall-Oriented Understudy for Gisting Evaluation (ROUGE), inference time, cost and task-specific JSON and conflict-detection measures. Pandas, NumPy, scikit-learn, sacrebleu, rouge-score and Matplotlib support the analysis. The line-level precision, recall and F1 implementation is locally defined and is retained for comparability with the earlier experiment, not presented as a published network-configuration metric. Recent peer-reviewed network-configuration research has proposed configuration-aware measures such as ConfigBLEU [7]. It will only be added if a faithful implementation can be reproduced.

## 5.9 Reproducibility and Version Control

Git and GitHub manage research code and revisions. The workflow records model and tokenizer identifiers, random seeds, dataset splits, package versions, CUDA and GPU

information, dataset checksums, training settings, adapters, runtime and result files. Trained adapters and relevant model artifacts will also be stored through the Hugging Face Hub.

### 5.10 Technical Environment Summary

The environment combines implemented dataset engineering and QLoRA fine-tuning with planned retrieval, automated correction and network validation within a reproducible Python and GPU-based workflow.

## Chapter 6: Ethical Considerations

This research evaluates Large Language Models for automated network configuration using publicly available datasets, technical documentation and controlled experimental environments. It does not involve human or animal participants, interviews, surveys, interventions or the collection of identifiable personal information; model performance will instead be assessed using predefined benchmarks and automated validation tools. The current study therefore does not involve activities requiring human research ethics approval under MIT policy. If personal data is identified or introduced during the research, the relevant ethical requirements will be reassessed before its use.

Ethical safeguards will nevertheless be maintained throughout the study. Datasets and technical materials will be screened for personal information, credentials and other sensitive content, while research files and model artifacts will be securely stored with access restricted to authorized users. Generated configurations will be evaluated in isolated, non-production experimental environments rather than deployed directly to operational networks, thereby reducing risks to production infrastructure.

Research integrity and transparency will also be maintained through accurate documentation of datasets, experimental procedures, model versions, evaluation methods and results, including findings that do not demonstrate performance improvements. The research will follow MIT's responsible research requirements and the principles of the Australian Code for the Responsible Conduct of Research. If the research later expands to include human participants, personal or private network information, or live operational environments, the ethical requirements will be reassessed and appropriate institutional approval obtained before such activities commence.

# Chapter 7: Initial Findings

This chapter presents the preliminary experimental work completed to date and the findings that guide the next stage of the research.

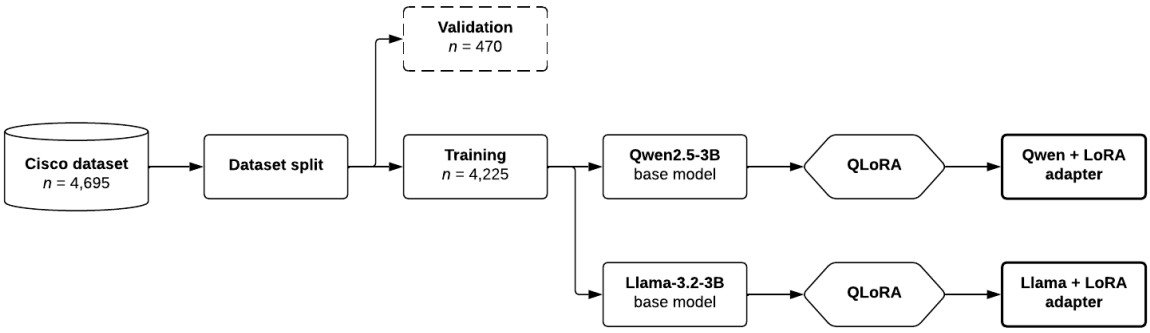
## 7.1 Scope of the Initial Experimental Investigation

The initial investigation examined two related questions. First, it tested whether useful network-configuration capability depends mainly on training examples that closely match the target Cisco IOS task, or whether broader networking supervision can also improve generation when the added data differ in vendor, task structure, or output format. Second, it compared Qwen2.5-3B-Instruct and Llama-3.2-3B-Instruct under the same adaptation approach to identify which model provided the stronger basis for subsequent experiments. The work therefore progressed from a Cisco-specific baseline to a broader multi-vendor, multi-task setting. The purpose was not to establish operationally deployable configuration generation at this stage, but to identify whether the models learned transferable configuration-related structure that justified deeper experimentation with retrieval, validation, and agentic correction.

## 7.2 Experimental Progression and Training Data

### 7.2.1 Cisco-Specific Baseline Experiment

The first experiment established a controlled Cisco-focused baseline. The source dataset was the requirement/question-to-configuration collection associated with SLM_netconfig [8]. It initially contained 5,096 Cisco records. After cleaning, removal of unusable examples, and an overlap check against the external evaluation collection, 4,695 records remained. These were divided into 4,225 training examples and 470 validation examples. Qwen2.5-3B-Instruct and Llama-3.2-3B-Instruct were fine-tuned separately with QLoRA using the same Cisco training data. This stage therefore asked a simple baseline question: when both small models receive only Cisco-oriented supervision, how effectively can they reproduce Cisco IOS configurations on an external set of requests?

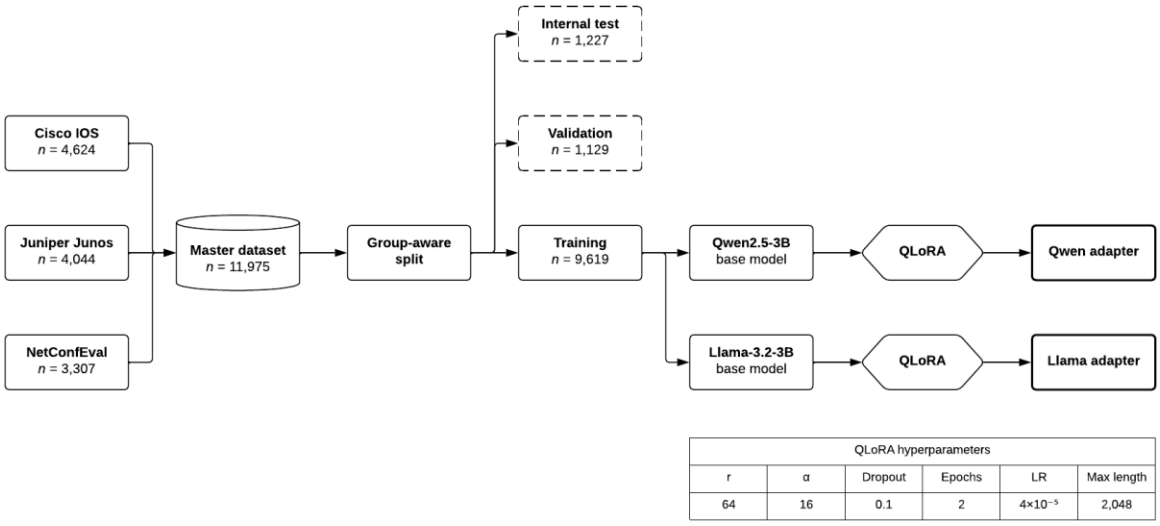



| QLoRA hyperparameters | | | | | |
|---|---|---|---|---|---|
| r | α | Dropout | Epochs | LR | Max length |
| 64 | 16 | 0.1 | 2 | $4\times10^{-5}$ | 384 |

Figure 7.1. Cisco-Specific Fine-Tuning Workflow

### 7.2.2 Multi-Vendor and Multi-Task Extension

The second experiment broadened supervision rather than merely enlarging the Cisco collection. The Cisco data were reprocessed using the standardized master-dataset cleaning procedure, reducing the retained records from 4,695 to 4,624. NetConfEval is itself a network-configuration benchmark covering structured networking tasks [1], but selected subsets were used here as training material rather than as the external Cisco evaluation benchmark. The combined dataset contained configuration generation, intent-to-JSON translation, and conflict-detection tasks. It was split into 9,619 training, 1,129 validation, and 1,227 internal-test records. Importantly, only 3,698 Cisco examples entered the second training split, fewer than the 4,225 Cisco examples used in the baseline experiment. The remaining supervision came from Juniper and NetConfEval, creating a useful setting for examining whether broader networking tasks could support performance on a Cisco-specific external benchmark.




QLoRA hyperparameters

| r | α | Dropout | Epochs | LR | Max length |
|---|---|---|---|---|---|
| 64 | 16 | 0.1 | 2 | $4\times10^{-5}$ | 2,048 |

Figure 7.2. Multi-Vendor and Multi-Task Fine-Tuning Workflow

## 7.3 Fine-Tuning Configuration and Computational Resources

This section summarizes the adaptation settings and computational resources used in the preliminary experiments.

### 7.3.1 Model Adaptation Setup

Both Qwen2.5-3B-Instruct and Llama-3.2-3B-Instruct were fine-tuned using QLoRA under broadly matched experimental settings, as summarized in Table 7.1. The principal difference between the two experimental stages was the maximum sequence length, which increased from 384 to 2,048 tokens for the multi-task dataset to accommodate longer NetConfEval examples. Consequently, the experiments are comparable in adaptation strategy and most training settings, but they should not be interpreted as a fully isolated ablation.

**Table 7.1. Fine-Tuning Settings**

| Parameter | Setting | Comparability |
|---|---|---|
| Method | QLoRA, 4-bit NF4 | Common |
| LoRA rank / alpha | 64 / 16 | Common |
| LoRA dropout | 0.1 | Common |
| Epochs | 2 | Common |
| Learning rate | $4 \times 10^{-5}$ | Common |
| Effective batch size | 16 | Common |
| Seed | 42 | Common |
| Target modules | Attention + feed-forward projections | Common |
| Maximum sequence length | 384 → 2,048 | Changed for multi-task data |

*Note: The sequence-length change is a genuine experimental difference and is treated as a limitation when interpreting causality.*

### 7.3.2 Computational Cost and Resource Usage

Table 7.2 summarizes recorded GPU time and approximate cost. Each four adapters cost under US$1.20 to train. The figures cover GPU compute for the model runs only and exclude other cloud overhead and development time.

Table 7.2. Training Resources and Cost

| Experiment | Model | GPU | Recorded runtime | Approx. GPU cost (USD) |
|---|---|---|---|---|
| Cisco-only | Qwen2.5-3B | A100-SXM4-80GB | 12.2 min | $0.32 |
| Cisco-only | Llama-3.2-3B | A100-SXM4-80GB | 10.7 min | $0.28 |
| Multi-task | Qwen2.5-3B | NVIDIA L40 | 86.61 min* | $1.14 |
| Multi-task | Llama-3.2-3B | NVIDIA L40 | 73.00 min* | $0.96 |

*Note: *Training plus validation time (Qwen: 84.41 + 2.20 min; Llama: 71.22 + 1.78 min). Costs use recorded rates of US$1.59 per GPU-hour for the A100 (RunPod) and US$0.79 per hour for the L40 (Thunder Compute). Values are approximate.*

## 7.4 Evaluation Design and Metrics

### 7.4.1 External Benchmark and Prompting Conditions

External evaluation used NetConfRAG-144, introduced as a curated dataset of natural-language networking intents for evaluating Cisco IOS intent-to-CLI generation [9]. This is distinct from all three training sources used in the multi-task dataset. NetConfRAG contains varied Cisco requests covering areas such as Dynamic Host Configuration Protocol (DHCP), access control, OSPF, policy-based routing, and related administrative tasks [9]. The raw file contained 144 rows. One corrupted duplicate-header row was removed, leaving 143 valid cases. Three fixed cases were reserved as in-context demonstrations, producing 140 scored questions. Each adapter was evaluated under zero-shot and three-shot prompting. Because the benchmark was designed for evaluation rather than training in this study, it provides a more meaningful test of transfer than measuring performance on the model's own held-out training-source records. However, complete semantic independence cannot be assumed until near-duplicate auditing against all training inputs is complete.

### 7.4.2 Evaluation Metrics

Four metric families were retained for the main analysis. Exact Match is the strictest measure: a prediction receives credit only when the complete generated configuration matches the reference after the defined evaluation processing. It therefore provides an easily interpretable measure of whole-output agreement, but it can penalize alternative configurations that may be functionally valid. Line F1 evaluates overlap at the command-line level by balancing how much of the generated configuration appears in the reference and how much of the reference is recovered. BLEU measures n-gram correspondence between generated and reference text and was originally developed for machine-translation evaluation [73]. ROUGE measures reference-oriented textual overlap [74]; ROUGE-L is particularly useful here because it captures longest-common-subsequence correspondence and therefore reflects ordered similarity across multi-line outputs. These metrics are complementary rather than interchangeable. High scores indicate stronger agreement with

the reference configuration, not proof that commands are syntactically accepted by a device or that the resulting network satisfies the intended policy. Operational correctness therefore requires separate validation.

## 7.5 Experimental Results

### 7.5.1 Overall Benchmark Performance

The matched external evaluation revealed a clear model-dependent response to broader supervision. Zero-shot Exact Match remained 0% for both Qwen and Llama with both adapters. Under three-shot prompting, Qwen improved strongly after multi-vendor and multi-task fine-tuning, whereas Llama did not and several of its similarity measures declined. This contrast is important: if Llama had been evaluated alone, the evidence would have suggested that broader supervision offered little benefit and may have weakened Cisco-specific generation. Evaluating both models therefore prevents a general conclusion from being drawn from one architecture.

Table 7.3. Three-Shot NetConfRAG Results

| Model | Training scope | Exact Match (%) | Line F1 | BLEU |
|---|---|---|---|---|
| Qwen | Cisco-only | 14.29 | 0.6939 | 49.41 |
| Qwen | Multi-task | 45.71 | 0.7814 | 67.66 |
| Llama | Cisco-only | 2.86 | 0.6934 | 42.93 |
| Llama | Multi-task | 3.57 | 0.6680 | 43.27 |

*Note: Results are from the recorded matched three-shot comparison on 140 scored NetConfRAG cases. Exact Match is reported as a percentage; the remaining similarity measures are aggregate scores.*

The case-level changes in Exact Match following multi-task fine-tuning are shown in Figure 7.3.

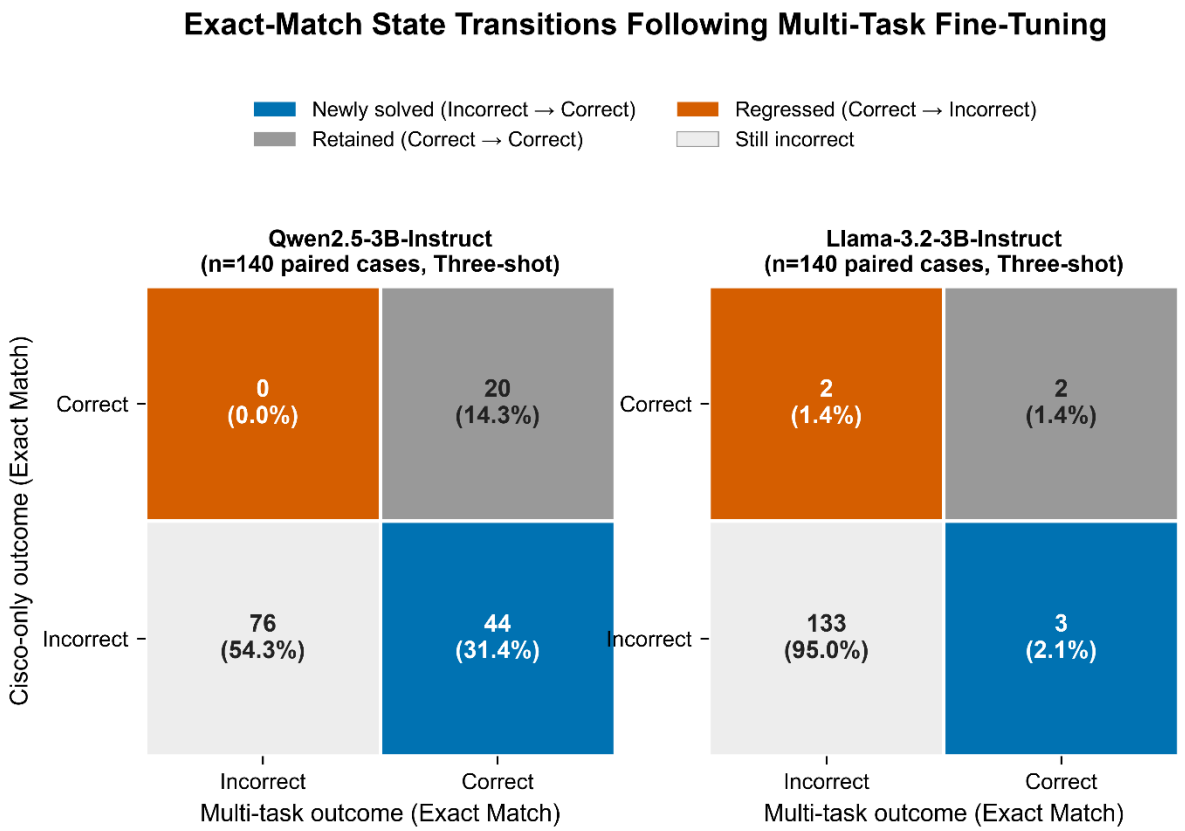



Figure 7.3. Exact-Match Transitions Following Multi-Task Fine-Tuning

Qwen gained 44 new exact matches with no regressions, while Llama gained three and lost two.

### 7.5.2 Qwen Performance

Qwen produced the central result of the experiment. With the Cisco-only adapter and three-shot prompting, Exact Match was 14.29%. With the multi-task adapter it increased to 45.71%, a gain of 31.42 percentage points and approximately 3.2 times the earlier exact-match rate. This difference is especially notable because the second model had been trained on fewer Cisco examples: 3,698 rather than 4,225. The improvement was not confined to exact string reproduction. Line F1 increased from 0.6939 to 0.7814, BLEU from 49.41 to 67.66, and ROUGE-L from 0.7730 to 0.8123. Line recall also rose more strongly than precision, indicating that the new adapter recovered a larger proportion of reference commands. Taken together, these measures show substantially stronger correspondence with complete and multi-line Cisco reference configurations under three-shot prompting.

### 7.5.3 Llama Performance

Llama produced a substantially different result from Qwen. Three-shot Exact Match increased only from 2.86% to 3.57%, while Line F1 declined from 0.6934 to 0.6680 and ROUGE-L fell from 0.7634 to 0.7414. BLEU increased only marginally, from 42.93 to 43.27. The broader dataset therefore did not improve Llama overall under this evaluation and weakened several measures of reference correspondence. Had Llama been studied alone, a reasonable preliminary interpretation would have been that multi-vendor and auxiliary-task supervision can dilute Cisco-specific performance. The opposing Qwen result shows why this effect must be tested rather than assumed.

### 7.5.4 Zero-Shot and Three-Shot Behavior

Prompting also interacted differently with the models. In zero-shot evaluation, Exact Match remained 0%, while Line F1 fell from 0.3610 to 0.3053 for Qwen and from 0.2334 to 0.1763 for Llama after multi-task training. Three demonstrations changed this picture for Qwen, whose Exact Match rose sharply, but Llama showed no comparable gain. Broader supervision was therefore not sufficient by itself, and the benefit of demonstration-based prompting was model-dependent.

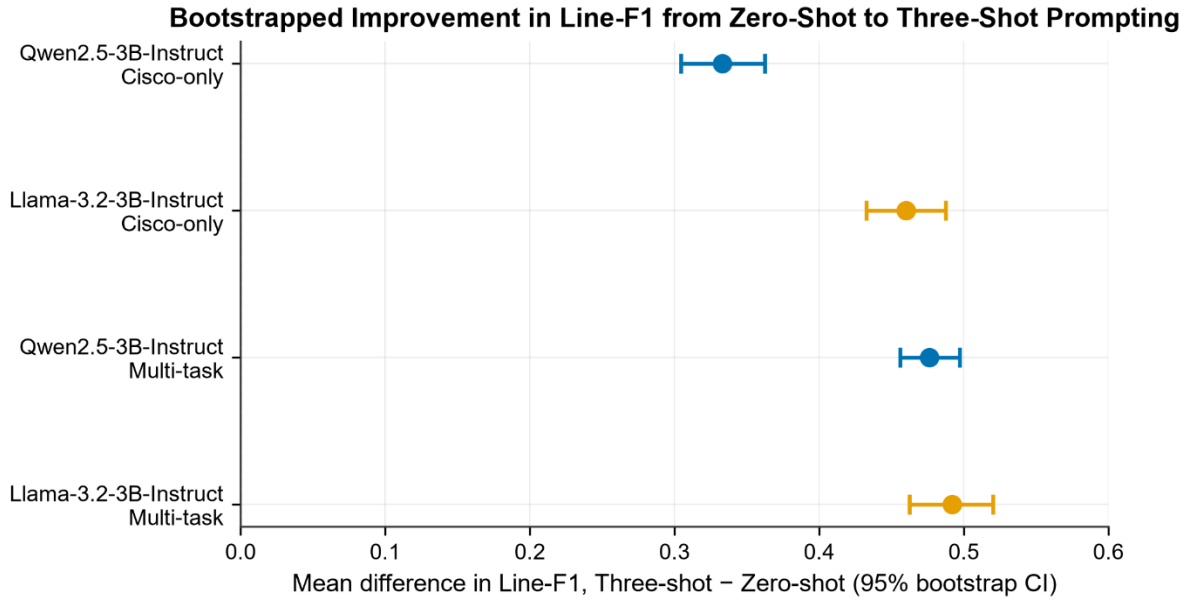



Figure 7.4. Effect of Three-Shot Prompting on Line-F1

Figure 7.4 shows that three-shot prompting produced a higher average Line-F1 than zero-shot prompting for every model and training condition. In all four comparisons, the 95%

confidence interval remains above zero, indicating that the average improvement is consistently positive rather than showing no change.

## 7.6 Principal Empirical Observation and Interpretation

This section interprets the main model-dependent findings and identifies the controlled comparisons required next.

### 7.6.1 Improvement Under Broader Training Supervision

The principal empirical observation is not simply that a larger dataset produced a better model. The same broader supervision was associated with sharply different outcomes across the two architectures. Qwen improved markedly even though its second training split contained fewer Cisco examples, while Llama showed almost no Exact Match gain and weaker Line F1 and ROUGE-L. Additional Cisco memorization is therefore an insufficient explanation, but broader training also cannot be assumed beneficial in general. The divergence suggests that reusable relationships between requirements, parameters, constraints, and structured outputs may be learned differently by different base models, making further controlled experimentation necessary.

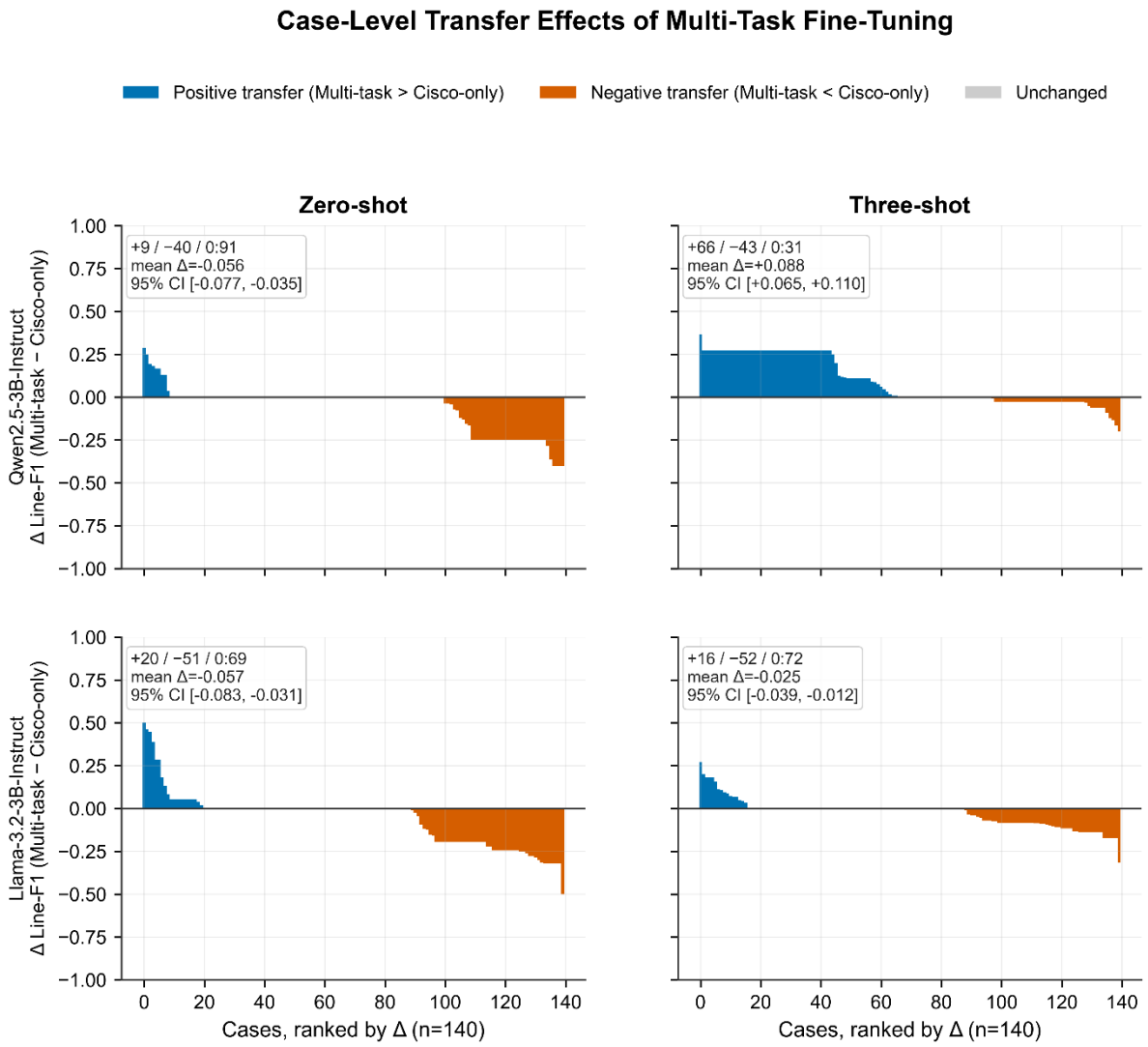



Figure 7.5. Case-Level Effects of Multi-Task Fine-Tuning

***Note.*** *+ indicates cases where multi-task fine-tuning improved Line-F1 relative to Cisco-only fine-tuning; − indicates cases where performance declined; 0 indicates no change. Mean Δ is the average Line-F1 difference across the 140 paired cases, calculated as multi-task minus Cisco-only. The 95% CI represents the bootstrap confidence interval around this mean difference.*

The case-level pattern reinforces the model-dependent response to broader supervision: under three-shot prompting, positive transfer dominates for Qwen, whereas Llama shows more negative than positive transfer.

### 7.6.2 Potential Cross-Vendor and Cross-Task Transfer

One plausible interpretation of the Qwen result is positive transfer across related networking tasks. Juniper examples differ syntactically from Cisco IOS but encode overlapping concepts such as interfaces, addressing, routing relationships, VLANs, and policy constraints. NetConfEval adds structured intent translation and conflict reasoning rather than simply more command strings. Such supervision may strengthen representations that later support Cisco generation. Llama, however, provides important counter-evidence to a universal transfer claim because the same broader dataset weakened several of its metrics. If only Llama had been studied, the preliminary interpretation could reasonably have been negative transfer. The opposing outcomes therefore indicate that any transfer effect is model-dependent.

### 7.6.3 Alternative Explanations and Need for Ablation

Several factors changed together: total training-set size, Cisco cleaning, conversational formatting, task labels, data composition, and maximum sequence length. These confounds prevent either the Qwen gain or Llama decline from being assigned to one cause. Their opposing responses make controlled ablation particularly important. Cisco-only, Cisco-plus-Juniper, Cisco-plus-NetConfEval, and complete multi-task training should therefore be compared under matched budgets and identical evaluation conditions.

## 7.7 Limitations and Interpretation Boundaries

Only two model families and two training epochs were evaluated, and the 11,975-record dataset remains limited compared with the range of Cisco IOS, Juniper Junos, and network-reasoning tasks. Although NetConfRAG was not used for supervised training, full benchmark independence requires near-duplicate checks, and the Cisco-focused evaluation does not yet demonstrate generalization to Juniper Junos, other vendors, or unseen task families.

## 7.8 Implications and Next Experimental Steps

**Controlled ablation:** The next experiments will compare Cisco-only, Cisco-plus-Juniper, Cisco-plus-NetConfEval and full multi-task training under matched conditions to determine whether the observed Qwen improvement is associated with cross-vendor supervision, auxiliary tasks, additional data volume or their combination.

**Operational validation:** Reference-based metrics will be complemented by configuration-aware validation using Batfish [71] and, where practical, virtual-device testing to assess syntax acceptance and intended network behavior.

## Chapter 8: Timeline for Research

The research program is organized across two twelve-week trimesters. Table 8.1 summarizes the present trimester, while the main planning emphasis is the feasibility of the subsequent experimental program.

### 8.1 Current Trimester

Table 8.1. Current Trimester Timeline

| Research activity | W1 | W2 | W3 | W4 | W5 | W6 | W7 | W8 | W9 | W10 | W11 | W12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scope, problem and research questions | | | | | | | | | | | | |
| Literature search and critical review | | | | | | | | | | | | |
| Research design and methodology | | | | | | | | | | | | |
| Dataset development and requirements | | | | | | | | | | | | |
| Preliminary fine-tuning | | | | | | | | | | | | |
| Preliminary evaluation and findings | | | | | | | | | | | | |
| Analysis and limitations | | | | | | | | | | | | |
| Report integration and revision | | | | | | | | | | | | |
| Feedback, referencing and submission | | | | | | | | | | | | |

### 8.2 Upcoming Experimental Trimester

The remaining experimental program is designed to be completed within the twelve-week trimester. Its feasibility is supported by the substantial progress already achieved, including dataset preparation, implementation of the fine-tuning pipeline, completion of preliminary experiments, and establishment of the evaluation workflow. Consequently, the next trimester extends an existing experimental framework rather than beginning implementation from the outset.

The schedule provides the largest allocation to fine-tuning and controlled experimentation, while later RAG and validation stages are restricted to selected models to contain computational demand. Activities are intentionally overlapped where they are independent, and limited time is retained for verification and necessary reruns. Continuous analysis and report writing further reduce end-of-trimester workload. Collectively, these measures provide sufficient scope to complete the planned experiments, final evaluation and thesis reporting within the available timeframe.

**Table 8.2. Experimental Timeline**

| Research activity | W1 | W2 | W3 | W4 | W5 | W6 | W7 | W8 | W9 | W10 | W11 | W12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Protocol and reproducibility checks | | | | | | | | | | | | |
| Dataset audit and evaluation freeze | | | | | | | | | | | | |
| Baseline evaluation | | | | | | | | | | | | |
| Fine-tuning and controlled ablations | | | | | | | | | | | | |
| Model-selection gate | | | | | | | | | | | | |
| Vendor-specific RAG development | | | | | | | | | | | | |
| RAG comparative experiments | | | | | | | | | | | | |
| Batfish validator integration | | | | | | | | | | | | |

| Research activity | W1 | W2 | W3 | W4 | W5 | W6 | W7 | W8 | W9 | W10 | W11 | W12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Validation and correction-loop experiments | | | | | | | | | | | | |
| Error, latency, cost and statistical analysis | | | | | | | | | | | | |
| Final benchmark and RQ synthesis | | | | | | | | | | | | |
| Report writing and revision | | | | | | | | | | | | |

## Chapter 9: Conclusion, Limitations and Future Work

### 9.1 Conclusion and Progress to Date

This research investigates how an open-weight Large Language Model (LLM) pipeline can translate natural-language network intents into accurate Cisco IOS and Juniper Junos configurations while remaining computationally efficient. The literature review showed that research in LLM-based network configuration is growing, but the evidence remains fragmented. Existing studies commonly use different models, datasets, vendors and evaluation methods, making direct comparison difficult. In particular, the review did not identify a controlled study that systematically compares the main improvement techniques under matched conditions and then uses the resulting evidence to determine the design of a final configuration-generation pipeline.

This gap defined the scope of the research and led directly to the research questions, objectives and experimental design. The study focuses on Cisco IOS and Juniper Junos configuration generation using selected open-weight models. The methodology follows three main stages. First, controlled experiments will examine how fine-tuning (FT) and Retrieval-Augmented Generation (RAG), individually and in combination, affect configuration accuracy and computational cost. Second, selected models will be evaluated within a validation and iterative-correction workflow to determine whether external feedback improves reliability and whether the improvement justifies its additional cost. Finally, the results from these

experiments will be combined to design an evidence-based pipeline that provides the strongest balance between accuracy and computational efficiency.

The preliminary work supports both the feasibility and the direction of this research. A multi-source dataset containing 11,975 prepared examples and the required fine-tuning and evaluation environment have already been developed. More importantly, the initial experiments showed substantially different responses between the two model families. Under the three-shot Cisco evaluation, Qwen Exact Match increased from 14.29% to 45.71% after broader training, whereas Llama changed only from 2.86% to 3.57%. These findings do not establish that broader training is generally superior. Instead, they demonstrate why controlled experimentation is necessary: the same broad intervention can produce very different outcomes across models.

Overall, the work completed to date establishes a clear research gap, a focused experimental approach and evidence that the proposed investigation is technically achievable. The remaining research will determine which techniques provide measurable improvements, whether those improvements are consistent across the selected models and vendors, and what computational cost they introduce. The final pipeline will therefore be designed from the experimental evidence rather than from assumptions about which techniques should perform best.

### 9.2 Limitations and Research Constraints

A major limitation is the availability and representativeness of domain-specific data. The research relies mainly on publicly available datasets, permitted technical documentation and controlled augmentation. These resources cannot represent the complete configuration knowledge available within network vendors or large operational organizations. Cisco, Juniper and network operators may possess substantially larger collections of validated configurations, historical changes, engineering rules and operational knowledge that are not publicly accessible. Consequently, the results obtained in this research should not be interpreted as representing the maximum performance achievable using richer proprietary datasets.

The study is also limited to Cisco IOS and Juniper Junos. These vendors provide an appropriate basis for controlled multi-vendor comparison, but they represent only part of the wider

networking ecosystem. Other vendors, operating systems, device families and software versions may produce different results. Even within Cisco and Juniper, the available dataset and benchmark cannot represent every protocol, command or configuration dependency. The findings must therefore remain bound to the vendors and task categories actually evaluated.

Model coverage creates another limitation. The study focuses primarily on selected open-weight Qwen and Llama models because they can be fine-tuned and evaluated repeatedly within the available computational budget. It is not feasible to evaluate every open-weight model family, parameter scale or proprietary frontier model using the complete experimental protocol. Larger models may behave differently and may respond differently to fine-tuning, RAG or validation. The study therefore aims to identify component effects under controlled conditions rather than establish a universal ranking of LLMs.

Computational constraints also affect model selection. Repeatedly fine-tuning very large models would require substantially greater GPU memory, runtime and financial resources. The investigation therefore prioritizes smaller and computationally practical models so that multiple controlled experiments can be conducted. This increases experimental feasibility but limits conclusions about whether the same effects would occur in substantially larger models.

Validation also has inherent boundaries. Batfish can provide useful evidence about supported configuration syntax and network behavior, but it cannot represent every vendor feature or operational condition. Passing the implemented validation checks cannot therefore be interpreted as proof that a configuration is universally safe for production deployment. Similarly, controlled or virtual environments cannot reproduce every characteristic of operational networks.

Finally, the thesis focuses specifically on configuration generation. LLMs and agentic AI could also contribute to troubleshooting, fault diagnosis, configuration-drift detection, monitoring, root-cause analysis and remediation. These tasks involve different data, tools and evaluation requirements and therefore remain outside the scope of the present investigation. The proposed agentic workflow should consequently be understood as bounded verifier-guided correction rather than unrestricted autonomous network operation.

### 9.3 Future work

Future research could extend this study by examining whether its findings remain consistent under broader data, model and vendor conditions. Collaboration with network vendors or operators, subject to appropriate anonymization and data-governance requirements, could provide access to richer operational configuration data than is available publicly. Synthetic data could also be investigated further where generated examples are retained only after suitable verification. Such work would help determine the extent to which the findings of this thesis depend on the characteristics and limitations of publicly available datasets.

The same controlled experimental framework could also be applied to additional model families and parameter scales. This would allow future studies to determine whether the contrasting behavior observed between Qwen and Llama persists across other architectures and whether the effectiveness of the investigated techniques changes as model capability and scale increase. Similarly, extending evaluation to additional network vendors, including held-out-vendor experiments, would provide stronger evidence about the generalizability of the findings beyond Cisco IOS and Juniper Junos.

The methodology could further be extended beyond configuration generation to related network-management tasks, including configuration repair, cross-vendor translation, troubleshooting and fault diagnosis using live network state. More advanced agentic architectures, including multi-agent systems, could also be examined, together with approaches that use verifier feedback during model training rather than only during inference-time correction.

Finally, future work should progressively examine operational validity through network emulation, evaluation with network engineers and carefully governed pilot deployments incorporating approval, monitoring and rollback mechanisms. These extensions would establish whether improvements demonstrated under controlled experimental conditions remain reliable in more complex and realistic network environments.

## Chapter 10: Plans for Future Publication

The work completed during the current stage of this research is intended to form the basis of an initial conference publication. A provisional title is **"*When Broader Fine-Tuning Helps One LLM but Not Another: Evidence from Network Configuration Generation.*"** The paper will focus on the experimental design and the contrasting responses observed between Qwen and Llama under broader multi-vendor and multi-task fine-tuning. The preliminary findings show a substantial improvement for Qwen, while Llama does not demonstrate a comparable gain under the same broader training direction. This model-dependent behavior provides the central motivation for further controlled investigation and forms a focused contribution suitable for early dissemination. The publication will also provide an opportunity to obtain feedback from the wider networking and AI research community and to support future research collaboration as the broader thesis develops.

## References


[1] C. Wang, M. Scazzariello, A. Farshin, S. Ferlin, D. Kostić, and M. Chiesa, "NetConfEval: Can LLMs Facilitate Network Configuration?," *Proc. ACM Netw.*, vol. 2, no. CoNEXT2, pp. 1–25, Jun. 2024, doi: 10.1145/3656296.

[2] E. Zeydan and Y. Turk, "Recent Advances in Intent-Based Networking: A Survey," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, Antwerp, Belgium: IEEE, May 2020, pp. 1–5. doi: 10.1109/VTC2020-Spring48590.2020.9128422.

[3] A. Mekrache, A. Ksentini, and C. Verikoukis, "Intent-Based Management of Next-Generation Networks: an LLM-Centric Approach," *IEEE Netw.*, vol. 38, no. 5, pp. 29–36, Sep. 2024, doi: 10.1109/MNET.2024.3420120.

[4] A. Leivadeas and M. Falkner, "A Survey on Intent-Based Networking," *IEEE Commun. Surv. Tutor.*, vol. 25, no. 1, pp. 625–655, 2023, doi: 10.1109/COMST.2022.3215919.

[5] J. Hong, N. V. Tu, and J. W. Hong, "A Comprehensive Survey on LLM-Based Network Management and Operations," *Int. J. Netw. Manag.*, vol. 35, no. 6, p. e70029, Nov. 2025, doi: 10.1002/nem.70029.

[6] R. Mondal, A. Tang, R. Beckett, T. Millstein, and G. Varghese, "What do LLMs need to Synthesize Correct Router Configurations?," in *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, Cambridge MA USA: ACM, Nov. 2023, pp. 189–195. doi: 10.1145/3626111.3628194.

[7] F. Li *et al.*, "PreConfig: A Unified Language Model Framework for Network Configuration Automation," *IEEE Trans. Cogn. Commun. Netw.*, vol. 12, pp. 6320–6330, 2026, doi: 10.1109/TCCN.2026.3662678.

[8] O. G. Lira, O. M. Caicedo, and N. L. S. D. Fonseca, "Network Self-Configuration Based on Fine-Tuned Small Language Models," *IEEE Open J. Commun. Soc.*, vol. 7, pp. 5920–5939, 2026, doi: 10.1109/OJCOMS.2026.3695460.

[9] Y. Ouni, K. Singh, A. Gourru, and F. Mhamdi, "WIP: Large Language Models for Network Automation: A Hierarchical Retrieval-Augmented Approach," in *2026 IEEE 27th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Bologna, Italy: IEEE, Jun. 2026, pp. 125–128. doi: 10.1109/WoWMoM69805.2026.00025.

[10] J. Wang *et al.*, "LLM-Powered Intent-Driven Configuration Generation for Multi-Vendor Networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 23, pp. 3537–3555, 2026, doi: 10.1109/TNSM.2026.3675409.

[11] D. Ficzere, G. Hollósi, and P. Varga, "Beyond Intent Translation: Research Gaps in the Application of Generative AI for Intent-Based Networking," in *NOMS 2025-2025 IEEE Network Operations and Management Symposium*, Honolulu, HI, USA: IEEE, May 2025, pp. 1–7. doi: 10.1109/NOMS57970.2025.11073701.

[12] "Cisco Crosswork Network Automation - Cisco Crosswork AI At a Glance," Cisco. Accessed: Sep. 17, 2026. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/networking/software/crosswork-network-automation/crosswork-ai-aag.html

[13] "Using the Mist MCP Server | Mist | Juniper Networks." Accessed: Sep. 17, 2026. [Online]. Available: https://www.juniper.net/documentation/us/en/software/mist/automation-integration/shared-content/topics/concept/about-juniper-mist-mcp.html

[14] K. Aykurt, A. Blenk, and W. Kellerer, "NetLLMBench: A Benchmark Framework for Large Language Models in Network Configuration Tasks," in *2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Natal, Brazil: IEEE, Nov. 2024, pp. 1–6. doi: 10.1109/NFV-SDN61811.2024.10807499.

[15] F. Li *et al.*, "ConfigTransLE: Large Language Models Enhanced Network Configuration Translation," *IEEE Trans. Netw.*, vol. 34, pp. 6380–6395, 2026, doi: 10.1109/TON.2026.3716703.

[16] H. Zaid, B. Shariati, P. Safari, J. K. Fischer, and R. Freund, "Empowering network management with a multi-agent LLM architecture," *J. Opt. Commun. Netw.*, vol. 18, no. 6, p. 568, Jun. 2026, doi: 10.1364/JOCN.585043.

[17] A. Vaswani *et al.*, "Attention Is All You Need," 2017, *arXiv*. doi: 10.48550/ARXIV.1706.03762.

[18] N. Tu, S. Nam, and J. W. Hong, "Intent-Based Network Configuration Using Large Language Models," *Int. J. Netw. Manag.*, vol. 35, no. 1, p. e2313, Jan. 2025, doi: 10.1002/nem.2313.

[19] R. Nakamura and K. Eguchi, "How Helpful Is LLM Assistance in Network Operations? a Case Study at a Large Demonstration Network," in *NOMS 2026-2026 IEEE Network Operations and Management Symposium*, Rome, Italy: IEEE, May 2026, pp. 1–10. doi: 10.1109/NOMS69089.2026.11668182.

[20] W. Enck *et al.*, "Configuration management at massive scale: system design and experience," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 3, pp. 323–335, Apr. 2009, doi: 10.1109/JSAC.2009.090408.

[21] A. S. Jacobs *et al.*, "Establishing Trust for Using Natural Language for Intent-Based Networking," *IEEE Trans. Netw. Serv. Manag.*, vol. 22, no. 5, pp. 4775–4787, Oct. 2025, doi: 10.1109/TNSM.2025.3574626.

[22] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "NAIL: A Network Management Architecture for Deploying Intent into Programmable Switches," *IEEE Commun. Mag.*, vol. 62, no. 6, pp. 28–34, Jun. 2024, doi: 10.1109/MCOM.001.2300313.

[23] M. Kiran, E. Pouyoul, A. Mercian, B. Tierney, C. Guok, and I. Monga, "Enabling intent to configure scientific networks for high performance demands," *Future Gener. Comput. Syst.*, vol. 79, pp. 205–214, Feb. 2018, doi: 10.1016/j.future.2017.04.020.

[24] X. Chang *et al.*, "KID: Knowledge Graph-Enabled Intent-Driven Network with Digital Twin," in *2022 27th Asia Pacific Conference on Communications (APCC)*, Jeju Island, Korea, Republic of: IEEE, Oct. 2022, pp. 272–277. doi: 10.1109/APCC55198.2022.9943779.

[25] Y. Dai, H. Zhang, J. Wang, J. Liao, and P. Zhang, "INCS: Intent-driven network-wide configuration synthesis based on deep reinforcement learning," *Comput. Netw.*, vol. 251, p. 110640, Sep. 2024, doi: 10.1016/j.comnet.2024.110640.

[26] H. Yu *et al.*, "Building a Comprehensive Intent-Based Networking Framework: A Practical Approach from Design Concepts to Implementation," *J. Netw. Syst. Manag.*, vol. 32, no. 3, p. 47, Jul. 2024, doi: 10.1007/s10922-024-09819-7.

[27] R. H. Ribeiro *et al.*, "A deterministic approach for extracting network security intents," *Comput. Netw.*, vol. 214, p. 109109, Sep. 2022, doi: 10.1016/j.comnet.2022.109109.

[28] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations," in *Proceedings of the 2016 ACM SIGCOMM Conference*, Florianopolis Brazil: ACM, Aug. 2016, pp. 328–341. doi: 10.1145/2934872.2934909.

[29] A. El-Hassany, P. Tsankov, L. Vanbever, and M. Vechev, "NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion," presented at the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 2018, pp. 579–594. Accessed: Oct. 02, 2026. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/el-hassany

[30] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A General Approach to Network Configuration Verification," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, Los Angeles CA USA: ACM, Aug. 2017, pp. 155–168. doi: 10.1145/3098822.3098834.

[31] A. Tang *et al.*, "Campion: debugging router configuration differences," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, Virtual Event USA: ACM, Aug. 2021, pp. 748–761. doi: 10.1145/3452296.3472925.

[32] Md. K. Hossain and W. Aljoby, "NetIntent: Leveraging Large Language Models for End-to-End Intent-Based SDN Automation," *IEEE Open J. Commun. Soc.*, vol. 6, pp. 10512–10541, 2025, doi: 10.1109/OJCOMS.2025.3642642.

[33] O. G. Lira, O. M. Caicedo, and N. L. S. Da Fonseca, "Large Language Models for Zero Touch Network Configuration Management," *IEEE Commun. Mag.*, vol. 63, no. 7, pp. 146–153, Jul. 2025, doi: 10.1109/MCOM.001.2400368.

[34] D. Donadel, F. Marchiori, L. Pajola, and M. Conti, "Can LLMs Understand Computer Networks? Towards a Virtual System Administrator," in *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, Normandy, France: IEEE, Oct. 2024, pp. 1–10. doi: 10.1109/LCN60385.2024.10639641.

[35] B. Ifland *et al.*, "GeNet: A Multimodal LLM-Based Co-Pilot for Network Topology and Configuration," in *2025 IEEE 45th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, Glasgow, United Kingdom: IEEE, Jul. 2025, pp. 117–122. doi: 10.1109/ICDCSW63273.2025.00026.

[36] J. Liu, L. Chen, D. Li, Y. Miao, and L. Ma, "Example Generalizing Network Configuration Synthesizer via Graph-Informed Large Language Models," *IEEE Trans. Netw.*, vol. 34, pp. 5191–5206, 2026, doi: 10.1109/TON.2026.3688476.

[37] A. Angi, A. Sacco, and G. Marchetto, "LLNet: An Intent-Driven Approach to Instructing Softwarized Network Devices Using a Small Language Model," *IEEE Trans. Netw. Serv. Manag.*, vol. 22, no. 4, pp. 3403–3418, Aug. 2025, doi: 10.1109/TNSM.2025.3570017.

[38] C. Provvedi, L. Seidenari, B. Picano, and R. Fantacci, "Intent-LLM: A Framework for Automated Network Configuration Through Code Generation," *IEEE Trans. Cogn. Commun. Netw.*, vol. 12, pp. 7246–7258, 2026, doi: 10.1109/TCCN.2026.3683230.

[39] Y. Dong, J. Sun, J. Li, and Y. Liu, "Learning to Configure Like Engineers: Manual-Guided Network Configuration Sketch Generation," *IEEE Trans. Netw. Serv. Manag.*, vol. 23, pp. 6351–6372, 2026, doi: 10.1109/TNSM.2026.3710600.

[40] S. Kou, C. Yang, and M. Gurusamy, "GIA: LLM-Enabled Generative Intent Abstraction to Enhance Adaptability for Intent-Driven Networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 11, no. 2, pp. 999–1012, Apr. 2025, doi: 10.1109/TCCN.2025.3545589.

[41] E. J. Hu *et al.*, "LoRA: Low-Rank Adaptation of Large Language Models," 2021, *arXiv*. doi: 10.48550/ARXIV.2106.09685.

[42] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," in *Advances in Neural Information Processing Systems 36*, New Orleans, Louisiana, USA: Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023, pp. 10088–10115. doi: 10.52202/075280-0441.

[43] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," 2020, *arXiv*. doi: 10.48550/ARXIV.2005.11401.

[44] K. B. Kan, H. Mun, G. Cao, and Y. Lee, "Mobile-LLaMA: Instruction Fine-Tuning Open-Source LLM for Network Analysis in 5G Networks," *IEEE Netw.*, vol. 38, no. 5, pp. 76–83, Sep. 2024, doi: 10.1109/MNET.2024.3421306.

[45] A. A. AlSamarneh, O. Y. Al-Jarrah, and A. T. Al-Hammouri, "NIT: A dataset for network intent translation," *Data Brief*, vol. 61, p. 111842, Aug. 2025, doi: 10.1016/j.dib.2025.111842.

[46] J. Wang *et al.*, "Hierarchical Index Retrieval-Driven Wireless Network Intent Translation With LLM," *IEEE Trans. Mob. Comput.*, vol. 24, no. 10, pp. 9837–9851, Oct. 2025, doi: 10.1109/TMC.2025.3564937.

[47] O. Ovadia, M. Brief, M. Mishaeli, and O. Elisha, "Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Miami, Florida, USA: Association for Computational Linguistics, 2024, pp. 237–250. doi: 10.18653/v1/2024.emnlp-main.15.

[48] A. Fogel *et al.*, "A General Approach to Network Configuration Analysis," presented at the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15),

2015, pp. 469–483. Accessed: Oct. 05, 2026. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/fogel
[49] J. Huang *et al.*, "Large Language Models Cannot Self-Correct Reasoning Yet," 2023, *arXiv*. doi: 10.48550/ARXIV.2310.01798.
[50] A. Madaan *et al.*, "Self-Refine: Iterative Refinement with Self-Feedback," in *Advances in Neural Information Processing Systems 36*, New Orleans, Louisiana, USA: Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023, pp. 46534–46594. doi: 10.52202/075280-2019.
[51] Z. Gou *et al.*, "CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing," 2023, *arXiv*. doi: 10.48550/ARXIV.2305.11738.
[52] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching Large Language Models to Self-Debug," 2023, *arXiv*. doi: 10.48550/ARXIV.2304.05128.
[53] M. Chen *et al.*, "Evaluating Large Language Models Trained on Code," 2021, *arXiv*. doi: 10.48550/ARXIV.2107.03374.
[54] I. Protogeros, R. Asadli, B. Hoffman, and L. Vanbever, "Benchmarking LLM-Driven Network Configuration Repair," 2026, *arXiv*. doi: 10.48550/ARXIV.2604.22513.
[55] R. Asadli, B. Hoffman, I. Protogeros, and L. Vanbever, "Evaluating Agentic Configuration Repair for Computer Networks," 2026, *arXiv*. doi: 10.48550/ARXIV.2606.06212.
[56] M. Scazzariello, L. Ariemma, and T. Caiazzi, "Kathará: A Lightweight Network Emulation System," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, Budapest, Hungary: IEEE, Apr. 2020, pp. 1–2. doi: 10.1109/NOMS47738.2020.9110351.
[57] "FRRouting User Guide — FRR latest documentation." Accessed: Oct. 05, 2026. [Online]. Available: https://docs.frrouting.org/en/stable-9.1/
[58] L. Zheng *et al.*, "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," in *Advances in Neural Information Processing Systems 36*, New Orleans, Louisiana, USA: Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023, pp. 46595–46623. doi: 10.52202/075280-2020.
[59] O. A. G. Lira, *oscarGLira/Fine-tuned-Configuration-Agent*. (Mar. 11, 2026). Jupyter Notebook. Accessed: Sep. 23, 2026. [Online]. Available: https://github.com/oscarGLira/Fine-tuned-Configuration-Agent
[60] Alaa Smarneh, "NIT." Hugging Face. doi: 10.57967/HF/3123.
[61] "NetConfEval/NetConfEval · Datasets at Hugging Face." Accessed: Oct. 05, 2026. [Online]. Available: https://huggingface.co/datasets/NetConfEval/NetConfEval
[62] Qwen *et al.*, "Qwen2.5 Technical Report," 2024, *arXiv*. doi: 10.48550/ARXIV.2412.15115.
[63] A. Grattafiori *et al.*, "The Llama 3 Herd of Models," Nov. 23, 2024, *arXiv*: arXiv:2407.21783. doi: 10.48550/arXiv.2407.21783.
[64] "Configuration file annotation — pybatfish 0.36.0 documentation." Accessed: Oct. 05, 2026. [Online]. Available: https://batfish.readthedocs.io/en/latest/config_annotation.html
[65] V. Zhong, C. Xiong, and R. Socher, "Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning," 2017, *arXiv*. doi: 10.48550/ARXIV.1709.00103.
[66] "Qwen/Qwen2.5-3B-Instruct · Hugging Face." Accessed: Oct. 05, 2026. [Online]. Available: https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
[67] "meta-llama/Llama-3.2-3B-Instruct · Hugging Face." Accessed: Oct. 05, 2026. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct
[68] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 9459–9474, 2020.
[69] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, 2019, pp. 3980–3990. doi: 10.18653/v1/D19-1410.
[70] J. Johnson, M. Douze, and H. Jegou, "Billion-Scale Similarity Search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, Jul. 2021, doi: 10.1109/TBDATA.2019.2921572.

[71] M. Brown, A. Fogel, D. Halperin, V. Heorhiadi, R. Mahajan, and T. Millstein, "Lessons from the evolution of the Batfish configuration analysis tool," in *Proceedings of the ACM SIGCOMM 2023 Conference*, New York NY USA: ACM, Sep. 2023, pp. 122–135. doi: 10.1145/3603269.3604866.

[72] "Where do I get IOS images? | GNS3 Documentation." Accessed: Oct. 05, 2026. [Online]. Available: https://mother.github.io/docs/troubleshooting-faq/where-do-i-get-ios-images

[73] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, Philadelphia, Pennsylvania: Association for Computational Linguistics, 2001, p. 311. doi: 10.3115/1073083.1073135.

[74] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. Accessed: Oct. 05, 2026. [Online]. Available: https://aclanthology.org/W04-1013/

## Appendices

The following appendices provide supporting technical, reproducibility and supplementary analytical material for the preliminary implementation reported in Chapters 4, 5 and 7. They are included to document the implemented workflow without interrupting the main argument of the candidature report. The appendices do not introduce additional research claims beyond those presented in the thesis.

## Appendix A: Reproducibility and Research Artefacts

### A.1 Research Code Repository

The implemented data-preparation, fine-tuning and evaluation workflow is maintained in a dedicated GitHub repository. The repository separates data engineering, model adaptation, evaluation outputs and documentation so that the preliminary experiments can be inspected and reproduced independently of the thesis document.

**GitHub repository:** https://github.com/Epetra1/llm-network-config-pipeline

***Table A.1. Primary Research Artefacts***

| Artefact | Location | Purpose |
|---|---|---|
| Source code and experiment repository | https://github.com/Epetra1/llm-network-config-pipeline | Data preparation, QLoRA fine-tuning, evaluation outputs, reports and experiment documentation. |
| Hugging Face account | https://huggingface.co/ChandraDhungana | Storage location for trained adapter artefacts and model-related outputs. |

### A.2 Repository Structure and Implemented Scope

The current repository covers the implemented stages of the proposal: dataset preparation, QLoRA fine-tuning and benchmark evaluation. Retrieval-Augmented Generation (RAG) and the agentic validation-and-correction loop remain planned stages for the next experimental

trimester and are therefore not represented as completed components in the present codebase.

```
llm-network-config-pipeline/
 data_preparation/
  juniper_augmentation/
  combined_dataset/
 fine_tuning/
  baseline_cisco_only/
  multitask/
   src/
 evaluation/
  netconfrag_140case/
 experiments_results/
 docs/
 requirements.txt
```

### A.3 Reproducibility Record

The implemented workflow records the experimental information required to reconstruct completed runs and distinguish one experimental condition from another. The reproducibility record includes:

- model and tokenizer identifiers;
- random seed and dataset partitions;
- package, CUDA and GPU information;
- dataset checksums and leakage-audit outputs;
- fine-tuning settings and saved adapters;
- runtime, cost and evaluation result files; and
- row-level and summary benchmark outputs used to generate the reported figures.

### A.4 Hugging Face Adapter Artefacts

Four adapter repository identifiers are associated with the implemented Cisco-only and multi-task experiments. They are listed here to preserve the mapping between the model family, training regime and stored adapter artefact.

***Table A.2. Adapter Repository Identifiers***

| Model | Training regime | Hugging Face repository ID |
|---|---|---|
| Qwen2.5-3B-Instruct | Cisco-only | ChandraDhungana/qwen2.5-3b-network-config-lora |
| Llama-3.2-3B-Instruct | Cisco-only | ChandraDhungana/llama3.2-3b-network-config-lora |
| Qwen2.5-3B-Instruct | Multi-task | ChandraDhungana/qwen2.5-3b-network-config-multitask-lora |
| Llama-3.2-3B-Instruct | Multi-task | ChandraDhungana/llama3.2-3b-network-config-multitask-lora |

*Access note: at the time this appendix was prepared, the Hugging Face profile did not expose public model repositories. If examiner access is intended, the repository visibility should be confirmed before final submission.*

## Appendix B: Dataset and Implemented Experimental Record

### B.1 Prepared Dataset Composition

The prepared multi-source dataset contains 11,975 retained records. The source-specific counts and roles used in the current implementation are summarized below.

***Table B.1. Prepared Dataset Composition***

| Source | Vendor / domain | Primary task role | Retained records |
|---|---|---|---|
| Cisco IOS dataset | Cisco IOS | Configuration generation | 4,624 |
| Juniper NIT | Juniper Junos | Configuration generation | 4,044 |

| | | | |
|---|---|---|---|
| NetConfEval | Generic / FRRouting-related tasks | Intent-to-JSON, conflict detection and selected configuration tasks | 3,307 |
| Combined dataset | Multi-source | Fine-tuning pool | 11,975 |

### B.2 Supervised Fine-Tuning Record Format

Each training record is converted into a conversation-style example. Source and provenance information are retained as metadata, while the model-facing messages explicitly identify the task, vendor, output type and input requirement.

***Table B.2. Example Fine-Tuning Record***

| Field | Example content |
|---|---|
| Record ID | CISCO_000123 |
| System message | You are a network automation assistant. Follow the requested task, vendor, and output format exactly. |
| User message | TASK: config_generation<br>VENDOR: cisco_ios<br>OUTPUT_TYPE: cli<br><br>INPUT:<br>Configure hostname R1. |
| Assistant target | hostname R1 |

### B.3 Partitioning and Leakage Controls

The combined dataset is partitioned reproducibly into training, validation and internal-test subsets. The principal controls used in the implemented data pipeline are:

- approximately 80% training, 10% validation and 10% internal testing;
- grouping of Juniper examples that share the same command-template lineage;
- grouping of Cisco records using normalized questions;

- task-specific splitting of NetConfEval records;
- cross-partition exact input-output duplicate checks; and
- similarity-based auditing to flag potentially overlapping examples for review.

For the current multi-task experiment, the resulting split contains 9,619 training records, 1,129 validation records and 1,227 internal-test records. The internal-test subset is excluded from training and model selection.

### B.4 Implemented QLoRA Configuration

Qwen2.5-3B-Instruct and Llama-3.2-3B-Instruct were adapted using broadly matched QLoRA settings. The maximum sequence length is the principal configuration difference between the Cisco-only and multi-task stages and is retained as an interpretation limitation.

***Table B.3. QLoRA Settings Used in the Preliminary Experiments***

| Parameter | Setting | Status across experiments |
|---|---|---|
| Method | QLoRA, 4-bit NF4 | Common |
| LoRA rank / alpha | 64 / 16 | Common |
| LoRA dropout | 0.1 | Common |
| Epochs | 2 | Common |
| Learning rate | $4 \times 10^{-5}$ | Common |
| Effective batch size | 16 | Common |
| Seed | 42 | Common |
| Target modules | Attention + feed-forward projections | Common |
| Maximum sequence length | 384 → 2,048 | Changed for multi-task data |

### B.5 External Evaluation Protocol

The preliminary external evaluation uses the NetConfRAG collection as a matched Cisco IOS intent-to-CLI benchmark. One corrupted duplicate-header row was removed from the 144-row file. Three fixed examples were reserved as in-context demonstrations, leaving 140 scored cases. Every adapter was evaluated on the same scored cases under zero-shot and three-shot prompting.

- 140 paired cases were scored under every model and training condition;
- zero-shot and three-shot prompting were evaluated;
- the compared models were Qwen2.5-3B-Instruct and Llama-3.2-3B-Instruct;
- the compared training regimes were Cisco-only and multi-task; and
- the reported reference metrics include Exact Match, line precision/recall/F1, BLEU and ROUGE.

## Appendix C: Supplementary Preliminary Results

This appendix presents supplementary analyses of the 140-case NetConfRAG evaluation. These figures are intentionally excluded from the main findings chapter to keep Chapter 7 focused on the central results. They provide additional descriptive evidence about error type, configuration-length sensitivity, score distributions and the relationship between evaluation quality and inference latency.

### C.1 Line-Level Error Profile

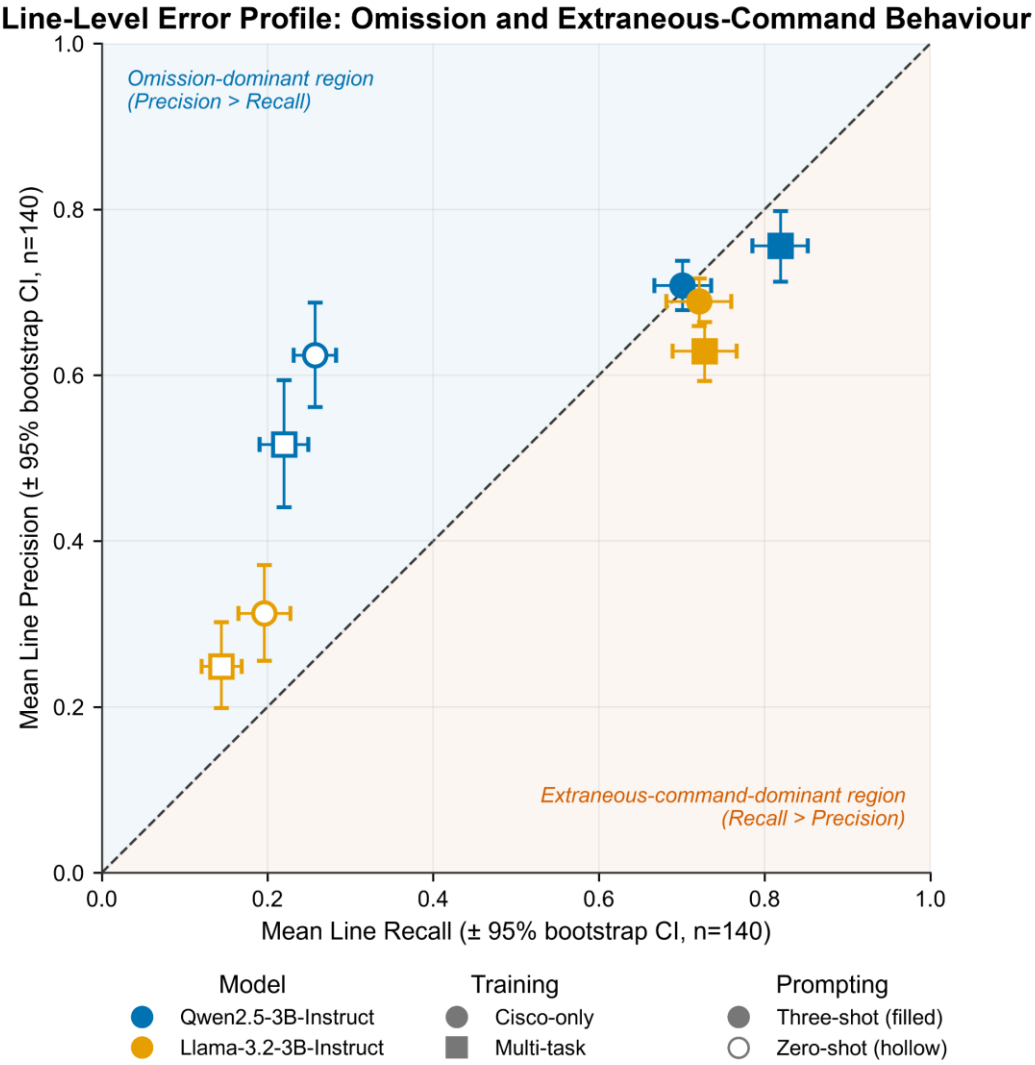



*Figure C.1. Line-Level Precision and Recall Error Profile*

*Note. Points above the diagonal are omission-dominant (precision exceeds recall), while points below the diagonal are extraneous-command-dominant (recall exceeds precision). Error bars show 95% bootstrap confidence intervals.*

The supplementary profile shows that zero-shot conditions are strongly omission-dominant for both model families. Three-shot prompting moves all conditions toward substantially higher recall and precision. The multi-task Qwen condition shows the strongest combined line-level performance among the displayed conditions, whereas the multi-task Llama condition remains lower and shifts toward a recall-greater-than-precision pattern.

### C.2 Sensitivity to Configuration Length

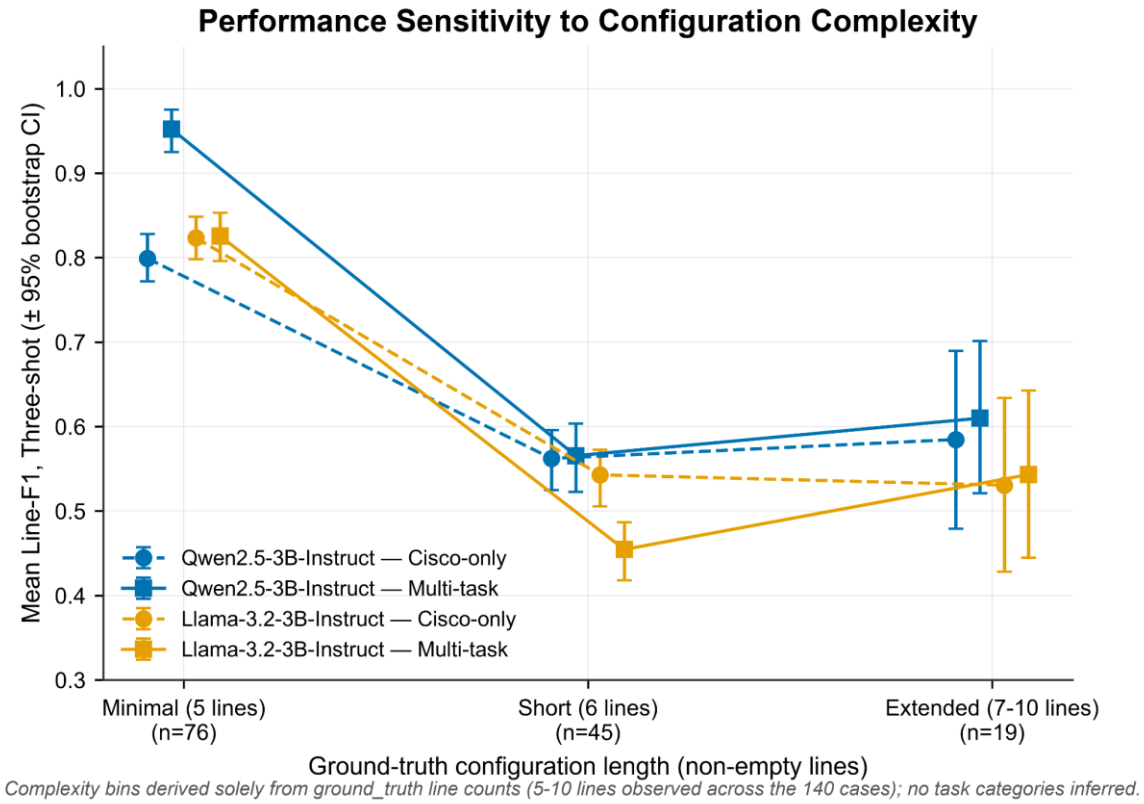



*Figure C.2. Performance Sensitivity to Configuration Complexity*

*Note. Complexity groups are defined only by the number of non-empty reference-configuration lines: minimal (5), short (6), and extended (7–10). The extended group contains fewer cases and therefore wider uncertainty intervals.*

Performance is highest for minimal five-line configurations and declines for the short group across all conditions. The extended group contains only 19 cases and shows wider confidence intervals, so it should be interpreted descriptively rather than as evidence of a monotonic complexity effect. The figure is useful for identifying configuration length as a factor to control in later benchmark design.

### C.3 Distribution of Per-Case Line-F1

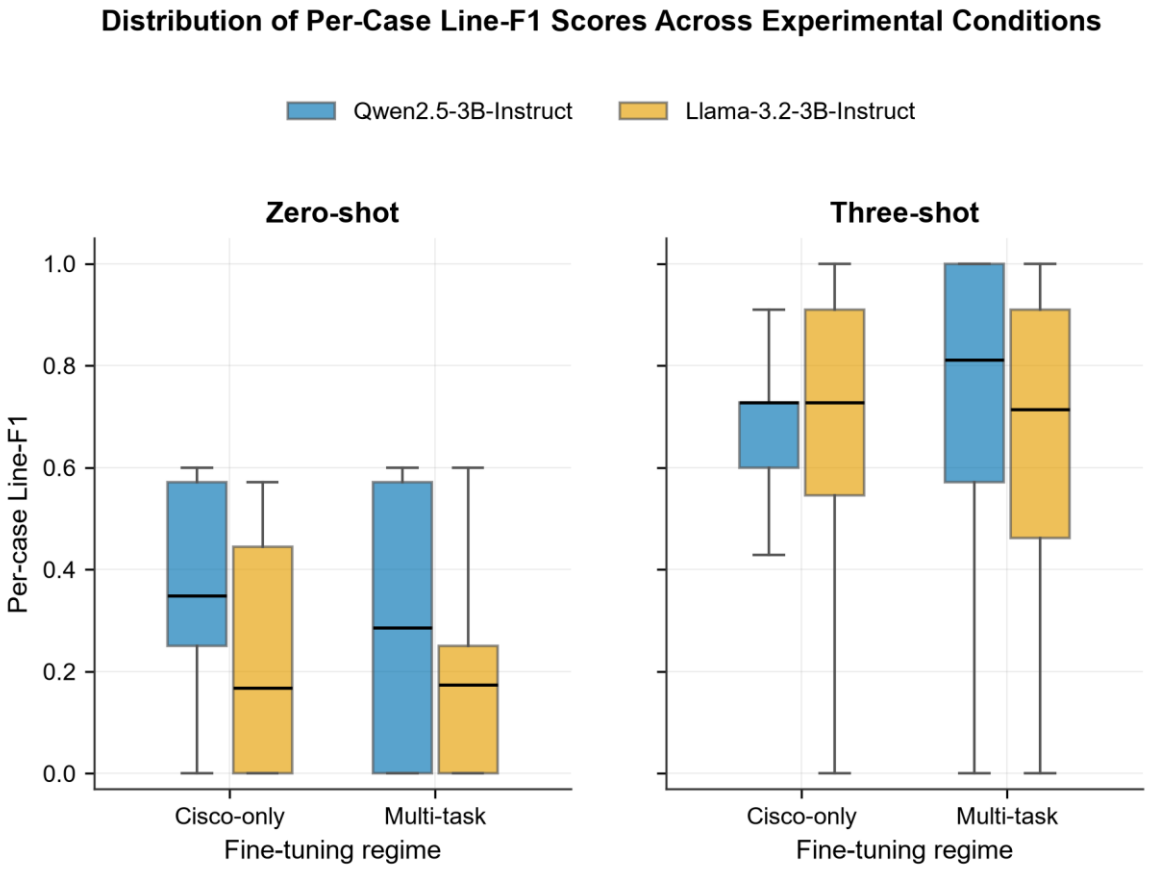



*Figure C.3. Distribution of Per-Case Line-F1 Scores Across Experimental Conditions*

*Note. Boxplots summarize per-case Line-F1 for the same 140 benchmark cases under zero-shot and three-shot prompting.*

The distributions reinforce the aggregate results without reducing them to a single mean. Three-shot prompting shifts the score distributions upward for both model families. Under three-shot prompting, Qwen with multi-task fine-tuning shows the strongest central tendency, while the corresponding Llama distribution does not show the same improvement.

## C.4 Accuracy-Latency Trade-Off

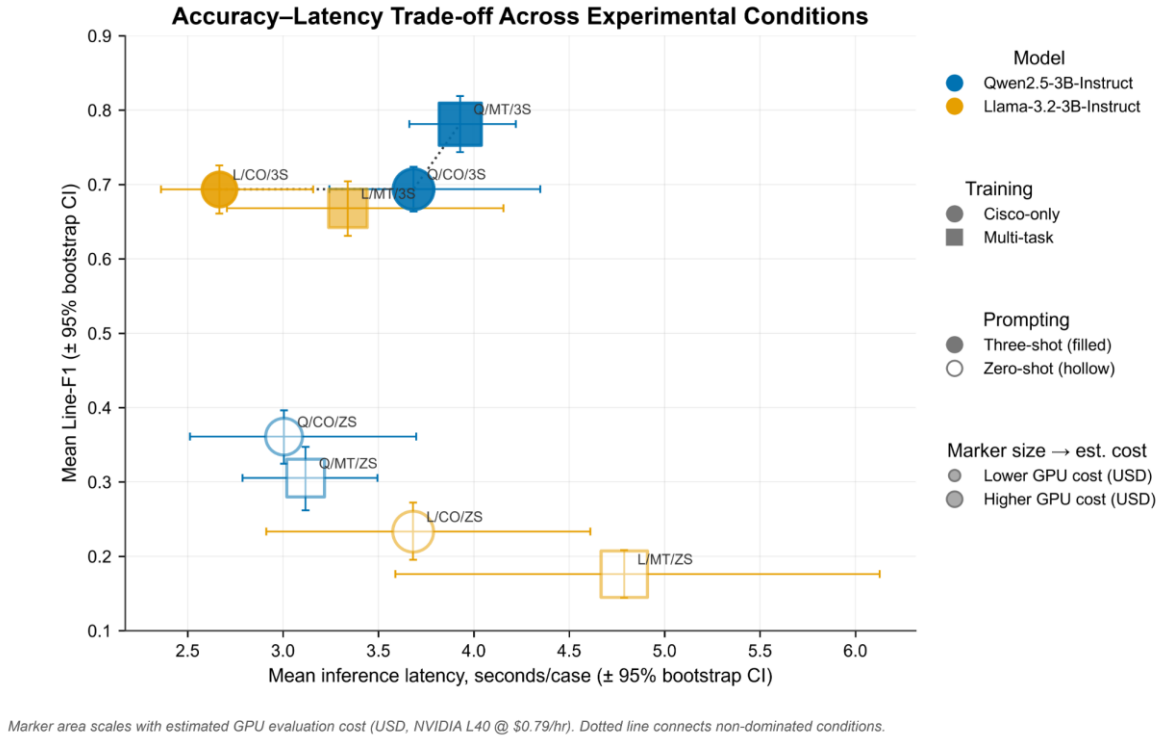



*Figure C.4. Accuracy-Latency Trade-Off Across Experimental Conditions*

*Note. Mean Line-F1 is shown against mean inference latency with 95% bootstrap confidence intervals. Marker size represents estimated evaluation GPU cost under the recorded L40 rate used for this analysis.*

The quality gain from three-shot prompting is accompanied by higher inference latency than the corresponding zero-shot conditions. Among the displayed preliminary conditions, multi-task Qwen under three-shot prompting provides the highest mean Line-F1 while remaining within the same general latency range as the other three-shot runs. This analysis is supplementary because the final thesis will evaluate computational cost together with syntactic and logical accuracy under the controlled pipeline experiments.

## Appendix D: Key Implementation Code Excerpts

This appendix presents selected, unmodified excerpts from the implemented project code. The listings were chosen because they provide direct evidence of the controls used in dataset preparation, leakage prevention, Juniper augmentation and QLoRA fine-tuning. Full source files remain available in the GitHub repository.

**Repository:** https://github.com/Epetra1/llm-network-config-pipeline

**Scope note:** The current repository implements data preparation, fine-tuning and evaluation. RAG and the agentic validation/correction loop remain planned work and are therefore not presented here as completed code.

| Listing | Implementation area | Reason for inclusion |
|---|---|---|
| D.1 | Group-aware dataset partitioning | Deterministic group-level train/validation/test assignment. |
| D.2 | Fuzzy leakage screening | TF-IDF cosine-similarity screening for possible partition leakage. |
| D.3 | Typed Juniper augmentation value generation | Typed, deterministic parameter substitution for Juniper augmentation. |
| D.4 | Centralized QLoRA experiment configuration | Single-source model, QLoRA and training hyperparameters. |
| D.5 | LoRA adapter construction | Validation of LoRA target modules and PEFT adapter creation. |

### D.1 Group-aware dataset partitioning

Shows the deterministic split logic that keeps each related group entirely within training, validation, or test data.

**Source file:** data_preparation/combined_dataset/scripts/prepare_final_dataset.py. The image below reproduces the selected source excerpt without rewriting the code.

**Code Listing D.1 - Group-aware dataset partitioning**

data_preparation/combined_dataset/scripts/prepare_final_dataset.py

```python
def assign_splits_by_group(group_ids: list, seed: int = SEED) -> dict:
    """Deterministically shuffles unique group ids and assigns each fully to
    one split. Returns {group_id: split_name}."""
    unique_groups = sorted(set(group_ids))
    rng = random.Random(seed)
    rng.shuffle(unique_groups)
    n = len(unique_groups)
    n_train, n_val, n_test = compute_split_sizes(n)
    mapping = {}
    for g in unique_groups[:n_train]:
        mapping[g] = "train"
    for g in unique_groups[n_train:n_train + n_val]:
        mapping[g] = "validation"
    for g in unique_groups[n_train + n_val:n_train + n_val + n_test]:
        mapping[g] = "test"
```

**Code Listing D.1. Group-aware dataset partitioning**

### D.2 Fuzzy leakage screening

Shows the TF-IDF cosine-similarity audit used to identify potentially overlapping inputs between training and validation/test partitions.

**Source file:** data_preparation/combined_dataset/scripts/prepare_final_dataset.py. The image below reproduces the selected source excerpt without rewriting the code.

**Code Listing D.2 - Fuzzy leakage screening**

data_preparation/combined_dataset/scripts/prepare_final_dataset.py

```python
def run_fuzzy_leakage_audit(master: pd.DataFrame, threshold: float = 0.85, top_k: int = 3):
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.metrics.pairwise import cosine_similarity
    import numpy as np
    suspicious = []
    for source in master["source_dataset"].unique():
        sub = master[master["source_dataset"] == source]
        train = sub[sub["split"] == "train"]
        test = sub[sub["split"].isin(["test", "validation"])]
        if len(train) < 2 or len(test) < 1:
            continue
        vect = TfidfVectorizer(min_df=1, max_features=20000)
        try:
            all_text = pd.concat([train["input"], test["input"]]).astype(str)
            tfidf = vect.fit_transform(all_text)
        except ValueError:
            continue
        train_mat = tfidf[: len(train)]
        test_mat = tfidf[len(train):]
        sims = cosine_similarity(test_mat, train_mat)
```

**Code Listing D.2. Fuzzy leakage screening**

### D.3 Typed Juniper augmentation value generation

Shows how new Juniper parameter values are generated according to the observed value type while excluding values already used in the row.

**Source file:** data_preparation/juniper_augmentation/augment_nit.py. The image below reproduces the selected source excerpt without rewriting the code.

**Code Listing D.3 - Juniper typed value generation**

data_preparation/juniper_augmentation/augment_nit.py

```python
def generate_value(rng, name, original_value, observed_pool_by_name, used_in_row):
    shape = value_shape(original_value)
    exclude = set(used_in_row) | {original_value}
    if shape == 'interface':
        return gen_interface(rng, exclude)
    if shape == 'ipv4':
        return gen_ipv4(rng, exclude)
    if shape == 'mac':
        return gen_mac(rng, exclude)
    if shape == 'priority_k':
        return gen_priority_k(rng, exclude)
    if shape == 'numeric':
        return gen_numeric(rng, name, original_value, exclude)
    if shape == 'quoted':
        return gen_quoted(rng, name, exclude)
    pool = observed_pool_by_name.get(name.lower(), [])
    return gen_word(rng, name, pool, exclude)
```

**Code Listing D.3. Typed Juniper augmentation value generation**

### D.4 Centralized QLoRA experiment configuration

Shows the centralized model identifiers, random seed, LoRA settings, 4-bit NF4 quantization and core training settings used by the multi-task experiment.

**Source file:** fine_tuning/multitask/src/config.py. The image below reproduces the selected source excerpt without rewriting the code.

**Code Listing D.4 - QLoRA experiment configuration**

fine_tuning/multitask/src/config.py

```
QWEN_MODEL_ID = "Qwen/Qwen2.5-3B-Instruct"
LLAMA_MODEL_ID = "meta-llama/Llama-3.2-3B-Instruct"
SEED = 42

LORA_R = 64
LORA_ALPHA = 16
LORA_DROPOUT = 0.1
CANDIDATE_LORA_TARGET_MODULES = [
    "q_proj", "k_proj", "v_proj", "o_proj",
    "gate_proj", "up_proj", "down_proj",
]
BNB_QUANT_TYPE = "nf4"
BNB_USE_DOUBLE_QUANT = True
BNB_LOAD_IN_4BIT = True

NUM_TRAIN_EPOCHS = 2
LEARNING_RATE = 4e-5
WEIGHT_DECAY = 0.001
MAX_GRAD_NORM = 0.3
EFFECTIVE_BATCH_SIZE = 16
LR_SCHEDULER_TYPE = "cosine"
WARMUP_RATIO = 0.03
```

**Code Listing D.4. Centralized QLoRA experiment configuration**

### D.5 LoRA adapter construction

Shows the implemented validation of model target modules and construction of the LoRA adapter before training.

**Source file:** fine_tuning/multitask/src/finetune.py. The image below reproduces the selected source excerpt without rewriting the code.

**Code Listing D.5 - LoRA adapter construction**

fine_tuning/multitask/src/finetune.py

```
valid_targets, missing_targets = get_valid_target_modules(model)
if missing_targets:
    print(f"[{short_name}] LoRA target modules not found (skipped): {missing_targets}")
if not valid_targets:
    raise RuntimeError(
        f"None of the candidate LoRA target modules {CANDIDATE_LORA_TARGET_MODULES} "
        f"exist in {short_name}'s module tree. Aborting before training."
    )
lora_config = LoraConfig(
    r=LORA_R, lora_alpha=LORA_ALPHA, lora_dropout=LORA_DROPOUT,
    target_modules=valid_targets, bias="none", task_type="CAUSAL_LM",
)
peft_model = get_peft_model(model, lora_config)
```

**Code Listing D.5. LoRA adapter construction**

### D.6 Reproducibility Interpretation

The code excerpts show that the preliminary work is supported by explicit implementation controls rather than only by reported outputs. Related examples are grouped before splitting, potential cross-partition similarity is audited, Juniper augmentation uses typed parameter generation, and QLoRA settings and adapter construction are defined directly in reusable source modules. These artefacts allow the implemented workflow to be inspected without placing full source files in the thesis body.

## Appendix E: Diagram Development Artefacts

All conceptual and workflow diagrams presented in this thesis were developed using Lucidchart and Figma. The editable source boards are provided below to support transparency, revision and reproducibility of the visual research artefacts.

**Table E.1. Diagram Development Artefacts**

| Tool | Use in Thesis | Editable Source |
|---|---|---|
| Lucidchart | Conceptual, methodological and workflow diagrams. | Lucidchart source board |
| Figma | Dataset-development and preparation diagrams and related visual layouts. | Figma source board |

**Lucidchart link:** https://lucid.app/lucidchart/ce4c9dbd-c28b-4641-a506-dba3236c24fd/edit?invitationId=inv_39598f15-db16-4ad0-aa5b-522e87495b37

**Figma link:** https://www.figma.com/board/pU7HPfJ7uWjOgv9fPeiL2R/Dataset-Development-and-Preparation?node-id=0-1&t=qptMD559hQ7nyGeR-1