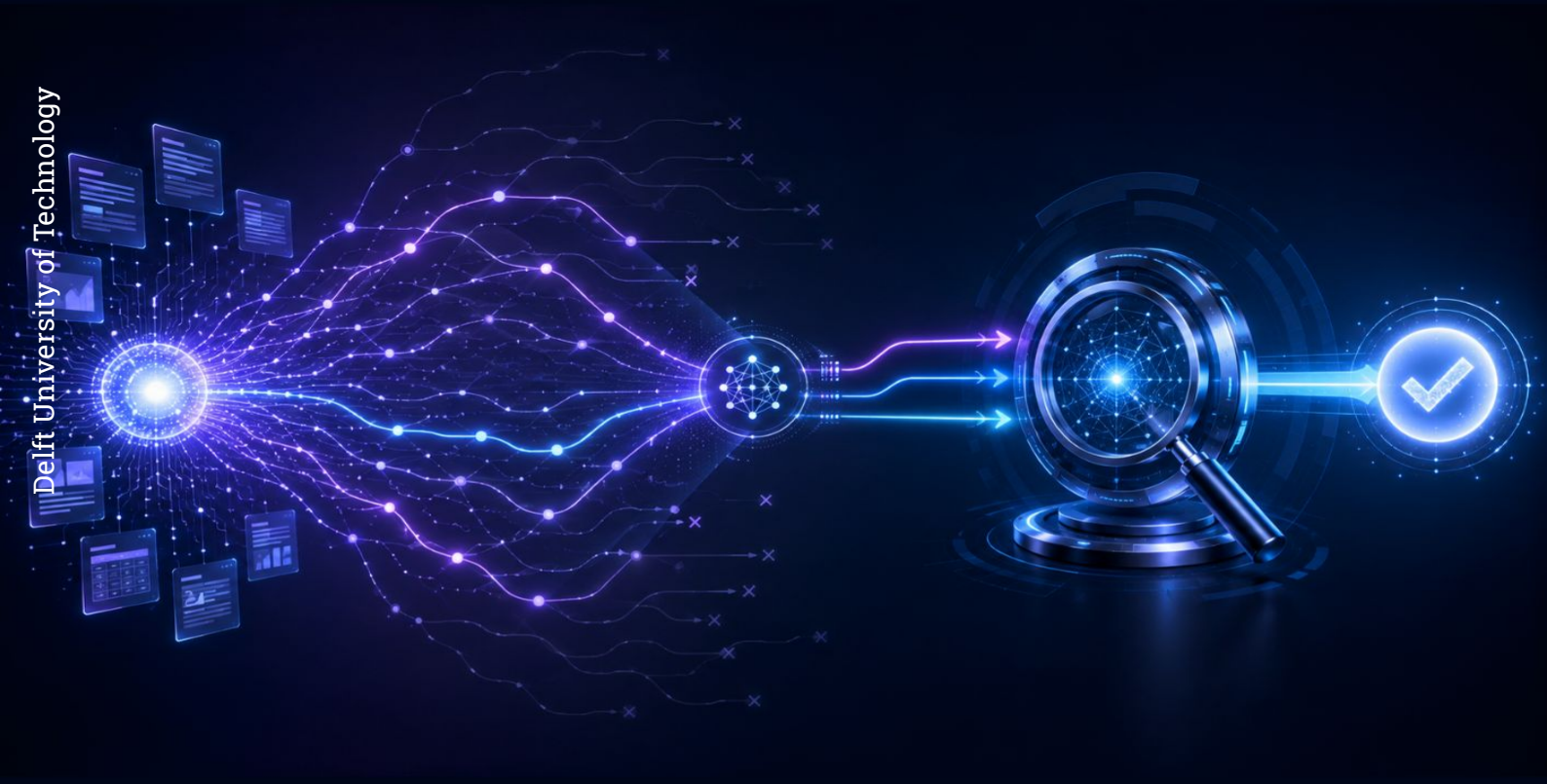
# Efficient Test-Time Scaling for Fact Checking with Large Language Models

Sowmya Prakash



Delft University of Technology

TU Delft

Cover image generated using ChatGPT

# Efficient Test-Time Scaling for Fact Checking with Large Language Models

THESIS

submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE**

in

**DATA SCIENCE AND**
**ARTIFICIAL INTELLIGENCE TECHNOLOGY**

by

**Sowmya Prakash**

| | |
|---|---|
| **Committee Chair:** | Dr. Avishek Anand |
| **First Core Member:** | Dr. Pradeep Murukannaiah |
| **Daily Supervisor:** | Dr. Venktesh Viswanathan |
| **Project Duration:** | November 2025 – June 2026 |
| **Faculty:** | Faculty of EEMCS, TU Delft |

Web Information Systems Group
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

# Preface

This thesis marks the completion of my Master's programme in Data Science and Artificial Intelligence Technology at Delft University of Technology. My interest in large language models, and in understanding how their reasoning capabilities can be used responsibly, led me to this research. Fact-checking real-world claims is an important and increasingly relevant problem, but building such systems is not only about improving accuracy. It is also about making them efficient enough to be practical when computational resources are limited.

Working on this thesis has been a challenging and meaningful journey. It came with moments of uncertainty, long nights, setbacks, and small victories, but most importantly, it brought immense growth. I leave this process knowing much more than I did when I started. Beyond the technical knowledge I gained, this thesis taught me patience, perseverance, and the value of continuing to learn from every stage of research, regardless of the outcome.

I would like to sincerely thank my supervisors, Dr. Avishek Anand and Dr. Venktesh Viswanathan, for their guidance, support, and encouragement throughout this thesis. Their thoughtful feedback and challenging questions helped me think more critically, refine my ideas, and approach the research process with greater clarity. I am deeply grateful for their mentorship and for the role they played in shaping both this thesis and my growth as a researcher.

I would also like to thank my parents, whose love, sacrifices, and constant support have made this journey possible. I am deeply grateful to my sister and to Ayush for their patience, encouragement, and belief in me throughout this period. Their support meant a great deal during the most demanding moments of this thesis.

Finally, I would like to thank my friends, family, and everyone who supported me along the way. Their encouragement helped me stay motivated and made this journey lighter and more memorable.

This thesis has been both demanding and rewarding, and I am grateful for everything it has taught me, both as a researcher and as a person.

***Sowmya Prakash***
*Delft, June 2026*

# Abstract

## Efficient Test-Time Scaling for Fact Checking with Large Language Models

**Abstract**

Large language models can support automated fact-checking by reasoning over claims and evidence, but single generated explanations can be unreliable for numerical, temporal, or complex claims. Verifier-guided Best-of-N (BoN) decoding improves performance by sampling multiple reasoning traces at inference time and selecting the trace most strongly supported by a verifier. However, exhaustive BoN applies the same fixed inference budget to every claim, making it costly even when a stable prediction emerges early. This thesis proposes a joint generation-and-verification optimization framework for dynamically allocating test-time computation in verifier-guided fact-checking through two methods. Verifier-Guided Adaptive Stopping uses verifier feedback as a confidence signal to stop generation once the prediction appears stable. Surrogate-Guided Selective Verification further uses an online linear surrogate model to estimate trace utility and prioritize which traces are sent for verification. A fixed verifier budget experiment then examines whether surrogate-guided allocation uses limited verifier calls more effectively than generation-order or random selection. Experiments on QuanTemp and ClaimDecomp show that adaptive stopping reduces inference latency and estimated cost by 42.0% and 43.7%, respectively, while maintaining performance comparable to exhaustive BoN. Surrogate-guided verification reduces verifier calls by 44.5% and 53.7% relative to exhaustive BoN, with additional reductions of 9.4% and 4.0% over adaptive stopping. Under limited verifier budgets, surrogate-guided allocation often outperforms generation-order and random selection at lower budget levels across the evaluated settings, showing that learned trace ordering can improve verification when verifier calls are constrained. These results show that adaptive generation and selective verification preserve much of the benefit of multi-trace reasoning while substantially reducing inference cost, and that fixed-budget surrogate allocation can prioritize useful traces when verifier calls are limited.

# Contents

# List of Tables

# List of Figures

# 1 Introduction

## 1.1. Background and Motivation

### 1.1.1. Fact-Checking in the Age of Large Language Models

The rapid growth of online information has made factual verification increasingly important. False and misleading claims can spread quickly across social media, news platforms, and other digital channels, influencing public understanding of topics such as politics, health, science, and finance [1, 2]. Although professional fact-checking provides high-quality verification, it is a labor-intensive process that requires identifying check-worthy claims, collecting relevant evidence, assessing source credibility, and justifying a verdict. The scale and speed of online information therefore motivates automated fact-checking systems that can support these stages computationally [3, 4, 5].

Large language models (LLMs) are promising for automated fact-checking because they can operate over natural language claims and evidence, generate intermediate explanations, and perform multi-step reasoning in a flexible manner. Prior work has explored neural and LLM-based approaches for several stages of the fact-checking pipeline, including evidence retrieval, veracity prediction, explanation generation, and claim decomposition [6, 7, 8, 9]. These capabilities make LLMs attractive for verifying real-world claims, where the task often requires more than lexical overlap between the claim and evidence.

At the same time, LLMs are not inherently reliable fact-checkers. They may produce fluent but unsupported explanations, rely on incomplete or noisy evidence, or make reasoning errors while presenting the final answer with high confidence [10]. This is particularly problematic in fact-checking, where plausibility alone is not sufficient: the verdict must be grounded in evidence, and the justification must support the predicted label. As a result, LLM-based fact-checking systems require mechanisms that can evaluate, compare, or aggregate generated reasoning before committing to a final prediction.

### 1.1.2. Complex and Numerical Claim Verification

Despite progress in automated fact-checking, real-world claims remain difficult because they are often not simple atomic statements. They may combine multiple factual aspects, require

evidence from several sources, or depend on intermediate reasoning steps before a final verdict can be assigned [11]. Benchmarks such as HoVer [12] and AVeriTeC [13] reflect this challenge by requiring systems to combine evidence across documents or question-answering steps. Work on claim decomposition further shows that complex claims often benefit from being broken into smaller verifiable questions [8, 11].

Numerical claim verification is a particularly challenging case. Claims involving quantities, dates, percentages, comparisons, or temporal intervals are sensitive to small changes in wording and values and minor inaccuracies can completely change the claim's veracity. QuanTemp highlights this challenge through real-world numerical claims involving comparative, statistical, interval, and temporal aspects [14]. Prior work also shows that language models still face limitations in numerical reasoning, especially when numerical information must be interpreted in context [15].

Together, complex and numerical claim verification motivate inference methods that go beyond a single generated answer. Since one reasoning path may overlook relevant evidence, fail to capture all aspects of a claim, or make a subtle numerical error, systems can benefit from generating multiple candidate reasoning traces and evaluating them before selecting a final verdict.

**[Claim]**: "Says in Newark "we're paying 80 percent of the school budget from local property taxes.""

**[Evidence snippets]**:

1. "Newark Public Schools operations are supplemented by $138.3 million from local property taxes, or 10.3% of the district's 2023–24 budget."
2. "In Newark's school budget this year, 86.3% of the district's funding comes from $1.2 billion in state aid."
3. "The local tax levy ... would remain unchanged from the current school year, at $138 million."

**[Verdict]**: REFUTES

**Figure 1.1:** Example from QuanTemp where the final verdict depends on a fine-grained comparison between the claim and evidence.

Figure 1.1 shows a QuanTemp example in which the verdict depends on identifying the relevant quantity in the evidence rather than matching words alone. The claim states that Newark pays 80% of its school budget from local property taxes. The evidence discusses the same school-budget context, but reports that local property taxes account for only 10.3% of the 2023-24 budget, while 86.3% comes from state aid. The claim is therefore refuted by a comparison between the claimed funding share and the evidence percentages. This illustrates why numerical claim verification requires evidence-grounded reasoning over quantities, entities, and context, not only lexical overlap between the claim and retrieved passages.

### 1.1.3. Test-Time Scaling and Verifier-Guided Reasoning

Scaling has been a central driver of progress in language models, with performance improving as model size, data, and training compute increase [16, 17]. However, reasoning performance can also be improved at inference time. Chain-of-thought prompting shows that generating intermediate reasoning steps helps LLMs solve complex tasks [18], while self-consistency further improves robustness by sampling multiple reasoning paths and aggregating their answers [19]. More recent work frames this broader idea as test-time scaling (TTS), where additional inference-time computation is used to search, sample, revise, or verify candidate outputs [20].

For fact-checking, this is useful because different reasoning traces may attend to different evidence or make different numerical comparisons. Verifier-guided Best-of-N decoding follows a generate-then-rank strategy: the system samples multiple candidate reasoning traces and uses a verifier to score their consistency with the claim and evidence before selecting a final prediction. This idea is related to prior work on verifier- and reward-model-guided reasoning, where a separate scoring model is used to rank candidate solutions produced by a generator [21, 22]. In this thesis, the same principle is applied to complex and numerical claim verification. As shown later in Table 5.1, Best-of-N decoding improves over top-1 decoding and self-consistency on the main verification metrics, suggesting that explicit trace ranking can be more effective than relying on a single generated trace or simple answer aggregation.

### 1.1.4. Computational Cost of Exhaustive Test-Time Scaling

While verifier-guided Best-of-N improves verification performance, it introduces a substantial computational cost. In the Exhaustive Best-of-N setting, the system samples a fixed number of reasoning traces for every claim and verifies all valid traces before selecting the final prediction. This means that each additional candidate requires both generation compute and, when verifier-guided selection is used, an additional verifier call. As a result, latency and cost increase with the number of sampled traces [20, 23].

The limitation is not only that fixed-budget scaling is expensive, but that it allocates computation uniformly. Increasing the trace budget gives every claim more reasoning attempts, regardless of whether the additional traces are likely to change the final verdict. This makes fixed-budget TTS a coarse scaling strategy. Some claims may produce a stable verifier preference after only a few traces, whereas more ambiguous claims may require additional exploration. Exhaustive Best-of-N treats both cases identically by spending the full trace and verification budget for every input.

Figure 1.2 extends this point from claim difficulty to budget allocation. Example A has relatively direct evidence for the central numerical claim, so additional reasoning traces may quickly become redundant. Example B is harder because the evidence contains near-matching quantities and related context, but not an exact match to the claim's numerical and temporal framing. This variation in claim difficulty makes uniform fixed-budget scaling poorly suited to real-world fact-checking.

**Example A: relatively direct evidence**

**[Claim]**: "Congressman (Ron) DeSantis sponsored legislation to increase sales taxes by 23 percent, hurting families, destroying jobs, devastating tourism."

**[Evidence snippets]**:

1. "In Congress, Ron DeSantis backed a national sales tax. A 23% tax hike ... DeSantis was a co-sponsor of the bill ..."
2. "2023-06-05 Gov. Ron DeSantis signed a $1.3 billion Florida sales tax relief bill into law. The legislation, which DeSantis describes as the largest tax ..."
3. "As a congressman, DeSantis also voted to increase the eligibility ages ... people who are really kind of against sales taxes state that it's a ..."

**Example B: hard claim with near-matching numerical evidence**

**[Claim]**: "We are looking at the USMCA, NAFTA 2.0 trade deal. That would be very important and would add a half a point of GDP and 180,000 new jobs per year if we get that through."

**[Evidence snippets]**:

1. "According to the U.S. International Trade Commission, USMCA will increase real GDP by $68.2 billion and create 176,000 new American jobs."
2. "The United States-Mexico-Canada Agreement (USMCA) entered into force on July 1, 2020. The USMCA, which substituted the North America Free Trade Agreement (NAFTA), is a mutually beneficial win for North American workers, farmers, ranchers, and businesses."
3. "According to the U.S. International Trade Commission (USITC), the agreement will have a positive and overarching impact on the economy."

**Figure 1.2:** Examples illustrating variation in claim difficulty. Highlighted spans show whether the evidence directly addresses the claim or is distributed across multiple factual aspects.

This cost-performance trade-off is visible in Table 5.3. On QuanTemp, Exhaustive Best-of-N requires 37,425 trace attempts and 37,098 verifier calls, resulting in 74,523 total operations. These results motivate the need for adaptive methods that preserve the benefits of verifier-guided test-time scaling while avoiding unnecessary generation and verification.

### 1.1.5. Adaptive Test-Time Scaling as a Cost-Performance Optimization Problem

The computational cost of Exhaustive Best-of-N raises a natural question: should every claim receive the same fixed inference budget, or can computation be allocated based on the difficulty of the claim? This question follows a broader trend toward adaptive inference, where systems dynamically decide when to retrieve, reason, or allocate additional computation based on the input or intermediate model confidence [24, 25]. Adaptive test-time scaling addresses this question by treating inference as a cost-performance optimization problem: the goal is not only to maximize verification performance, but also to minimize unnecessary generation, verification, latency, and compute cost.

In verifier-guided fact-checking, this trade-off is especially important because compute is spent in two places. The generator must produce candidate reasoning traces, and the verifier must

score them before a final prediction can be selected. Exhaustive Best-of-N assumes that all claims require the same number of traces and verifier calls, even though some claims may produce a stable label preference early. This suggests that adaptive methods could stop generation when additional traces are unlikely to change the final verdict, or reduce verification when cheaper signals are sufficient to guide the decision. A diagnostic analysis of the Exhaustive BoN trace pools supports this intuition. On QuanTemp, a trace with the gold label appears within the first five trace attempts for 72.5% of claims. A similar pattern appears on ClaimDecomp, where 71.5% of claims contain a gold-label trace within the first five attempts. These statistics suggest that useful traces often appear well before the full fixed budget is exhausted, motivating adaptive stopping as a way to avoid unnecessary generation.

Beyond deciding when to stop generating new traces, there is also a verifier-allocation problem. In verifier-guided Best-of-N, every valid generated trace is typically sent to the verifier model, even though some traces may be low-value, redundant, or unlikely to affect the final prediction. If cheaper trace-level signals can identify which candidates are most worth verifying, the system may reduce verifier usage without relying on exhaustive scoring. This motivates studying not only Adaptive Stopping, but also selective verification under limited verifier budgets. A fixed-budget setting further isolates this question by asking how performance changes when only a limited number of generated traces can be verified.

This thesis therefore studies adaptive test-time scaling as a practical inference-time optimization problem for complex and numerical claim verification. The core gap addressed in this work is that verifier-guided Best-of-N decoding improves prediction quality by spending more inference-time computation, but it does not decide whether that computation is necessary for a particular claim. A principled adaptive approach should preserve the benefits of multiple reasoning traces while stopping trace generation or reducing verifier usage when additional computation is unlikely to change the final verdict. The central question is therefore whether fact-checking can be made more efficient while maintaining verification performance close to the exhaustive verifier-guided baseline.

## 1.2. Research Questions

This thesis investigates whether verifier-guided test-time scaling for complex and numerical claim verification can be made more efficient without substantially reducing verification quality. The work is guided by the following research questions:

**RQ1:** To what extent can Verifier-Guided Adaptive Stopping improve fact-checking efficiency while maintaining performance comparable to exhaustive verifier-guided Best-of-N decoding?

**RQ2:** To what extent can Surrogate-Guided Selective Verification further reduce verifier usage by jointly optimizing reasoning trace generation and verifier allocation?

**RQ3:** How does the allocation of verifier calls affect performance in verifier-guided test-time

scaling under fixed verifier budgets?

## 1.3. Scientific Contribution

The contributions of this work, derived from addressing the proposed research questions, are as follows:

1. **Verifier-Guided Adaptive Stopping for claim verification.** This thesis proposes Verifier-Guided Adaptive Stopping, a margin-based adaptive test-time scaling strategy for generate-then-verify claim verification. Instead of assigning every claim a fixed trace budget, the method uses verifier-score-based feedback to stop reasoning trace generation once the predicted label appears sufficiently stable.
2. **Cost-aware analysis of generate-then-verify inference.** This work evaluates verifier-guided test-time scaling as a cost-performance problem rather than only as a prediction problem. It separately measures reasoning trace generation, verifier calls, latency, token usage, total operations, and estimated GPU energy cost, making the trade-off between verification quality and inference cost explicit.
3. **Surrogate-Guided Selective Verification.** This thesis introduces Surrogate-Guided Selective Verification, which uses an online linear surrogate to estimate verifier scores from inexpensive trace-level features. The surrogate is used to prioritize which generated traces should receive verifier scoring, enabling selective verifier allocation within a multi-trace fact-checking pipeline.
4. **Fixed-budget verifier allocation analysis.** This work separately studies the surrogate as a trace-selection mechanism under fixed verifier budgets. This analysis isolates the question of how limited verifier calls should be allocated across an existing candidate pool, independent of the full end-to-end generation cost.
5. **Evaluation across verifier choices and inference budgets.** The proposed adaptive strategies are evaluated under different verifier configurations and inference budgets, including larger candidate trace pools. This analysis examines how verifier choice, trace budget, and allocation strategy affect the trade-off between cost, latency, and verification quality.
6. **Empirical study on numerical and decompositional claim verification.** The methods are evaluated on QuanTemp and ClaimDecomp, which provide numerical and decompositional claim verification settings. These evaluations show how adaptive test-time scaling behaves in tasks that require quantitative, temporal, compositional, and evidence-grounded reasoning.

## 1.4. Thesis Outline

The report is laid out as follows. Chapter 2 discusses related work on fact-checking, test-time scaling, adaptive inference, and selective evaluation. Chapter 3 introduces the generate-

then-verify framework, the Exhaustive Best-of-N baseline, Verifier-Guided Adaptive Stopping, and Surrogate-Guided Selective Verification. Chapter 4 describes the datasets, models, implementation details, experimental configurations, fixed-budget verifier-allocation setup, and evaluation metrics used to answer the research questions. Chapter 5 presents the empirical results for the baseline comparison, Adaptive Stopping, Surrogate-Guided Selective Verification, fixed-budget verifier allocation, and ablation studies. Finally, Chapter 6 summarizes the main findings and discusses limitations, assumptions, future work, and AI disclosure.

# 2

# Related Work

## 2.1. Fact Checking and Verification

Automated fact-checking is formulated as a claim verification task where the system determines the verdict of the claim as supported, refuted or not enough evidence, based on retrieved set of evidence [26]. Many early systems follow a three-stage pipeline of document retrieval, evidence selection, and final label prediction, often using natural-language inference models [27]. While effective for simple claims, such pipelines struggle with complex numerical or political claims, which require aggregating multiple evidence sources and multi-hop reasoning [12, 28]. To address this, [8] break complex claims into explicit and implied sub-questions that can be answered from the available evidence before making a final veracity judgement. This decomposition strategy is particularly effective for political claims which often combine multiple explicit statements with underlying assumptions or meanings.

Emerging fact-checking pipelines leverage test-time scaling by performing multi-step chain-of-thought (CoT) reasoning [19, 29, 30]. These systems typically generate multiple reasoning traces per query and aggregate them using mechanisms such as majority voting or confidence-based selection, which empirically improves accuracy and robustness [18, 31, 19]. Although multi-step CoT generally improves performance, the generated reasoning is still often flawed or internally inconsistent [32, 33], which makes verification a crucial step. Combining CoT reasoning with external verification at inference time has become a key component of reliable fact-checking systems [34, 35, 36]. The verifiers or reward models score candidate solutions and intermediate steps to catch reasoning flaws and reduce hallucinations.

Recent work has also begun to consider the efficiency of verification itself. MiniCheck [23] fine-tunes small models on synthetically generated grounding examples to perform sentence-level fact-checking, achieving similar performance at a fraction of the inference cost. LiLaVe [37] takes the approach of training lightweight latent verifiers that score candidate outputs directly in the model's representation space, avoiding the need for full autoregressive generation during verification. Both works treat verification as a scoring task rather than a generative one, and both highlight efficiency as an increasingly important research direction when verification must be applied repeatedly or at scale.

## 2.2. Test-Time Scaling for Reasoning Tasks

Test-Time Scaling (TTS) improves the reasoning capabilities of large language models by allocating additional computation at inference time. Methods such as Chain-of-Thought [18] and Tree-of-Thought [38] prompting encourage models to generate explicit intermediate reasoning steps, improving performance on complex tasks. When multiple reasoning traces are sampled, aggregation strategies such as self-consistency [19] and graph-based decoding [39] select the most reliable answer from the candidate set, substantially improving accuracy on numerical, multi-hop, and document-level benchmarks.

Learned reward models offer an alternative to aggregation, where reasoning traces are scored. *Outcome reward models* (ORMs) [21] evaluate the final answer or verdict produced by a trace, while *process reward models* (PRMs) [40, 22] assess the quality of individual intermediate reasoning steps, providing finer-grained feedback than surface-level answer agreement. In verifier-guided Best-of-N (BoN) selection, multiple traces are sampled and the one with the highest reward model score is selected as the final prediction rather than relying on majority voting. [29] adopts this approach for fact-checking, where a reward model can be trained to score the consistency between a generated justification and its proposed verdict.

A key finding of [20] is that test-time compute can be more effective when allocated through search or verifier-guided selection rather than through naive increases in the number of sampled traces. This shifts the scaling question from whether more inference-time computation helps to how that computation should be allocated. However, many TTS methods still rely on fixed inference budgets [41], spending the same amount of computation on every input regardless of difficulty.

Recent work has also highlighted diminishing returns and "overthinking" effects, where increasing test-time computation leads to longer or more complex reasoning chains that add cost and latency without reliably improving, and sometimes even degrading, the final answer [42, 43, 44]. These findings suggest that inference-time scaling is not only a question of increasing compute, but also of deciding when additional computation is useful.

## 2.3. Adaptive and Budgeted Inference

Adaptive computation has been widely explored in broader machine learning literature through early-exit architectures [45], which aim to reduce average inference cost by dynamically terminating computation once sufficient confidence is reached, often via auxiliary classifiers [46, 47], and anytime prediction frameworks, which use progressive refinement strategies in predictive modelling [48].

Within LLMs, recent work explores adaptive reasoning through confidence-based decisions for invoking chain-of-thought reasoning [49] and token-budget-aware decoding [50], which constrains the number of thinking tokens based on predicted difficulty. [41] propose self-calibration as a mechanism for allocating test-time compute, routing easy inputs to cheaper

inference paths and harder inputs to longer reasoning chains. These methods reduce computation within a single model invocation but do not address how to allocate computation across multiple model calls in a pipeline.

A complementary line of work considers adaptive decisions across multi-step inference pipelines. In retrieval-augmented generation (RAG), [25] train a classifier to decide whether a given query requires single-step or multi-step retrieval based on its complexity, while [24] formulate retrieval as an active decision made iteratively as generation proceeds rather than as a fixed pre-processing step. In fact-checking, [29] similarly studies adaptive decomposition, where claims are decomposed selectively based on estimated complexity, and Best-of-N selection for choosing among generated verdicts. Closer to the verification setting, [51] study budget-aware test-time scaling with discriminative verification, showing that selectively applying a verifier rather than scoring all candidates can preserve accuracy while substantially reducing inference cost. [52] similarly find that collaborative verification across multiple reasoning traces improves accuracy without proportionally increasing compute, provided that verification is applied selectively.

## 2.4. Bandit-Inspired Selective Evaluation

Multi-armed bandits provide a classic framework for sequential decision making under uncertainty, where a learner repeatedly selects among a set of actions and observes a reward for the chosen action [53]. The core challenge is the exploration-exploitation trade-off where the learner must balance selecting actions with high observed rewards against exploring uncertain actions that may yield better outcomes. Common selection strategies include greedy selection [54], which always picks the action with the highest current estimate, and optimistic strategies such as Upper Confidence Bound (UCB) [55], which adds an uncertainty bonus to encourage exploration of less-evaluated actions.

Linear bandits extend this framework to settings where each action is described by a feature vector and rewards are assumed to be linear in those features [56]. Rather than maintaining independent estimates per action, the learner fits a shared linear model that generalises reward estimates across actions based on feature similarity, allowing feedback from evaluated arms to inform predictions for related unevaluated ones. Best-arm identification [57] is a practically important variant that focuses on finding the single highest-reward action within a fixed budget of evaluations, rather than minimising cumulative regret across many rounds. This objective is natural when a fixed pool of candidates must be ranked and only the top-scoring one is ultimately selected.

While linear bandits provide efficient and interpretable reward estimates, they can be limited when rewards depend on non-linear interactions between features. Non-linear contextual bandit methods, including kernel-based [58] and neural bandit approaches [59], extend this setting by learning richer reward functions while still using uncertainty-aware selection rules. These methods are useful when simple feature-reward relationships are insufficient, but they typically

introduce additional computational and modelling complexity.

Several bandit variants study settings where feedback is costly or actions are consumed upon evaluation. Disposable linear bandits [60] constrain each arm to a limited number of selections, motivated by recommendation scenarios where repeated exposure is undesirable. [61] apply challenger-arm sampling to efficient in-context reasoning, framing the choice of which reasoning examples to evaluate as a bandit problem and using observed utility to avoid scoring low-value candidates. Online relevance estimation [62] applies bandit-based selective scoring to large-scale document retrieval, using cheaper surrogate estimates to defer expensive relevance model calls until they are most informative.

## 2.5. Relevance to This Thesis

The reviewed literature situates this thesis at the intersection of verifier-guided fact checking, test-time scaling, and adaptive budgeted inference. Existing fact-checking pipelines show that external verification is important for improving the reliability of generated reasoning traces, especially when chain-of-thought reasoning can be inconsistent or hallucinated. At the same time, verifier-guided test-time scaling methods demonstrate that sampling multiple reasoning traces and selecting among them can improve final performance, but these approaches often apply fixed inference budgets and rely on repeated calls to expensive verifier or reward models.

Prior work on adaptive inference and bandit-based selective evaluation motivates a more dynamic view of inference-time computation. Adaptive computation methods show that not all inputs require the same amount of processing, while bandit and best-arm identification frameworks study how limited feedback opportunities can be allocated selectively across candidate actions. Related retrieval work, such as online relevance estimation, further shows that expensive scoring models can be applied selectively while cheaper estimates guide which candidates should be evaluated next. Together, these works motivate studying fact-checking pipelines in which verification is treated as a limited resource and inference-time computation is allocated adaptively rather than uniformly.

# 3
# Methodology

## 3.1. Problem Setup

This thesis studies claim verification with retrieved evidence. For each input claim $c$, the system receives a set of evidence passages

$$E = \{e_1, e_2, \ldots, e_k\}.$$

The goal is to predict a veracity label for the claim using the provided evidence. The label is expected to belong to the claim verification label space:

$$y \in \{\text{Supports}, \text{Refutes}, \text{Conflicting}\}.$$

Here Supports indicates that the evidence supports the claim, Refutes indicates that the evidence contradicts the claim, and Conflicting indicates that the evidence is mixed, incomplete, or internally inconsistent with respect to the claim.

In addition to the final label, the system produces a natural-language justification. A candidate prediction can therefore be represented as a pair $(j_i, y_i)$, where $j_i$ is the generated justification and $y_i$ is the predicted label. This representation is useful as claim verification errors could often be reasoning errors. A prediction may use the correct evidence but draw the wrong conclusion, or it may predict a plausible label while relying on a weak or unsupported justification.

## 3.2. Generate-Then-Verify Framework

The proposed framework treats claim verification as a test-time scaling (TTS) problem. Instead of relying on a single generated explanation, the system uses additional inference-time compute to sample multiple reasoning traces for a claim-evidence pair. This is useful because different samples may focus on different evidence passages, resolve numerical or temporal details differently, or produce alternative label interpretations.

Generating more traces alone, however, does not guarantee better verification. While additional samples can increase diversity, they may also introduce unsupported explanations or labels that are inconsistent with the evidence. Therefore, the framework separates generation

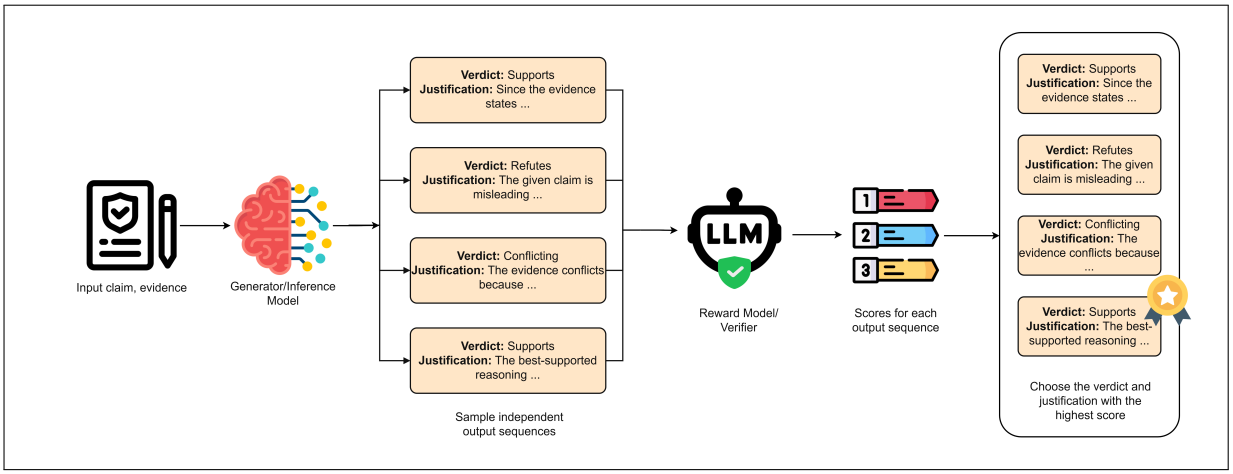



**Figure 3.1:** Overview of the generate-then-verify framework for verifier-guided Best-of-N selection.

from selection: a generator first samples multiple candidate reasoning traces, and a verifier then scores them to select the most reliable prediction.

Given a claim $c$ and evidence set $E$, the generator produces:

$$T = \{t_1, t_2, \ldots, t_n\},$$

where each trace $t_i$ contains a justification $j_i$ and a predicted label $y_i$.

Traces without a valid label are excluded from verifier-based selection, but are still counted in efficiency analysis. Each valid trace is scored by the verifier:

$$s_i = V(c, y_i, j_i),$$

where $s_i$ measures how consistent the claim, predicted label, and justification are.

The final prediction is selected by choosing the label associated with the highest verifier score:

$$\hat{y} = y_{\arg\max_i s_i}.$$

Figure 3.1 gives the overview of generate-then-verify pipeline and Figure 3.2 gives a concrete example of this framework. It shows how different sampled traces can propose different labels for the same claim, and how the verifier score is used to select the final verdict from the valid generated traces.

This shared generate-then-verify formulation forms the basis for all experiments in this thesis. The exhaustive baseline verifies every valid trace under a fixed budget. Verifier-Guided Adaptive Stopping terminates generation once verifier confidence and label agreement are sufficiently strong. Surrogate-Guided Selective Verification further reduces verifier usage by estimating verifier scores and selectively verifying only the most promising traces.

**[Claim]**: 'The non-partisan Congressional Budget Office concluded ObamaCare will cost the U.S. more than 800,000 jobs."

**[Evidence Passages]**:

1. '... earlier this year, the head of the congressional budget office testified before congress that implementation of obamacare would cost 800,000 jobs ..."
2. '... congressional budget office testified before congress that implementation of obamacare would cost 800,000 jobs ..."
3. '... yes, obamacare will eliminate some 800,000 jobs ..."

**[Gold Label]**: CONFLICTING

**[Trace 1]**

- **Predicted Label**: CONFLICTING
- **Verifier Score**: 0.987
- **Justification**: The evidence does not fully support the claim. While it mentions a projection of 800,000 jobs, it does not confirm 'more than" 800,000. Other statements introduce ambiguity and inconsistency, leading to a conflicting interpretation.

**[Trace 2]**

- **Predicted Label**: REFUTES
- **Verifier Score**: 0.153
- **Justification**: The evidence states that implementation would cost 800,000 jobs, which contradicts the claim's wording of 'more than 800,000". This numerical mismatch is used to refute the claim.

**[Trace 3]**

- **Predicted Label**: SUPPORTS
- **Verifier Score**: 0.043
- **Justification**: The evidence is interpreted as supporting the claim by citing testimony about job loss estimates, although it does not strictly justify the 'more than" qualifier.

**[Final Verdict]**: CONFLICTING (highest verifier score)

**Figure 3.2:** Example generate-then-verify outcome showing candidate traces, verifier scores, and final verdict selection.

## 3.3. Exhaustive Best-of-N Baseline

The first experimental setting uses an exhaustive test-time scaling strategy. For each claim, the system samples up to $N = 15$ reasoning traces and verifies every trace that contains a valid predicted label. This setting is used as the baseline because it represents the full-budget version of the proposed generate-then-verify framework: no valid trace is skipped, and no adaptive decision is made to reduce computation. It therefore provides a fixed-budget reference point for comparing prediction quality and inference-time cost.

The motivation for this baseline is that multiple sampled traces can improve robustness by exploring different reasoning paths or focusing on different parts of the evidence. Verifying all valid traces gives the system the largest opportunity to identify a strong justification-label pair,

but it also incurs the highest verifier cost among the proposed methods.

In this baseline, the shared generate-then-verify procedure from Section 3.2 is applied directly. Invalid parsed outputs are excluded from verifier-based selection, and the final prediction is chosen using the same verifier-guided Best-of-N rule, i.e., the label of the trace with the highest verifier score.

The resulting accuracy, latency, number of verifier calls, generator tokens, and total runtime cost serve as a reference point for evaluating whether later adaptive methods can preserve prediction quality while reducing computation.

Algorithm 1 formalizes the exhaustive baseline. It highlights that generation uses a fixed trace budget and that every valid labeled trace is sent to the verifier before the final prediction is selected.

**Algorithm 1** Exhaustive Best-of-N Baseline

**Require:** Claim $c$, evidence set $E$, fixed trace budget $N = 15$
**Ensure:** Final verdict $\hat{y}$
Construct prompt $p$ from claim $c$ and evidence set $E$
Initialize valid trace set $\mathcal{T}_{valid} \leftarrow \emptyset$
**for** $i = 1$ to $N$ **do**
    Generate reasoning trace $t_i$ using prompt $p$
    Parse justification $j_i$ and label $y_i$ from $t_i$
    **if** $y_i \in \{\text{Supports}, \text{Refutes}, \text{Conflicting}\}$ **then**
        Compute verifier score $s_i = V(c, y_i, j_i)$
        Add $(t_i, j_i, y_i, s_i)$ to $\mathcal{T}_{valid}$
    **end if**
**end for**
**if** $\mathcal{T}_{valid} = \emptyset$ **then**
    **return** missing-label prediction
**end if**
Select $i^\star = \arg\max_i s_i$ over $\mathcal{T}_{valid}$
**return** $\hat{y} = y_{i^\star}$

## 3.4. Verifier-Guided Adaptive Stopping

The exhaustive baseline verifies all valid traces up to a fixed budget, regardless of what has already been learned from earlier verifier calls. This treats the trace budget as a constant rather than as a decision that can be informed by the model's own verification feedback. Adaptive Stopping is introduced as a more principled alternative: instead of deciding the number of traces in advance, it uses verifier scores observed during inference as a confidence signal for whether further generation is still necessary.

The main motivation is that the verifier provides more than a final ranking over traces. Its scores also indicate how strongly the current generated evidence supports one label relative to the alternatives. If one label has already received a substantially higher verifier score than competing labels, additional traces are less likely to change the final decision. Conversely,

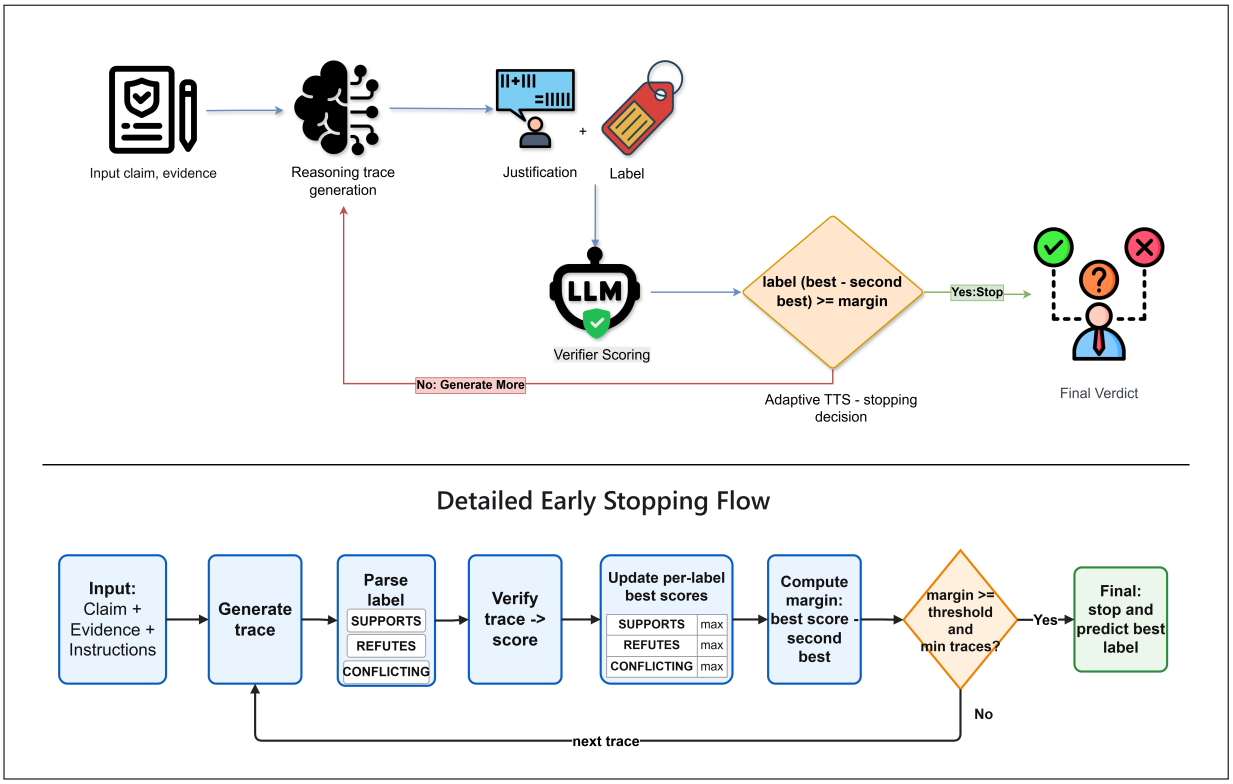



**Figure 3.3:** Overview of Verifier-Guided Adaptive Stopping, where traces are generated and verified until the label-level margin stopping criterion is satisfied.

when verifier scores remain close across labels, the claim is treated as uncertain and the system continues generating traces up to the maximum budget.

Adaptive Stopping keeps the same generate-then-verify framework defined in Section 3.2, but adds a stopping rule after each valid verified trace. The goal is not to change the final selection rule, but to decide dynamically whether the current verifier feedback is strong enough to stop. In this way, inference compute is allocated according to the observed difficulty of each claim: easier claims can terminate early, while ambiguous claims continue to use more of the available budget.

The stopping rule is based on a label-level verifier margin. For each observed label $y$, the system keeps the highest verifier score assigned to any verified trace predicting that label:

$$b_y = \max_{i:y_i=y} s_i,$$

where $y \in \mathcal{Y} = \{\text{Supports}, \text{Refutes}, \text{Conflicting}\}$. The labels are then ranked by these best scores. If $y_{(1)}$ and $y_{(2)}$ are the labels with the highest and second-highest best scores, the margin is

$$m = b_{y_{(1)}} - b_{y_{(2)}}.$$

A larger margin indicates that the verifier currently prefers one label clearly over the strongest alternative.

**Algorithm 2** Adaptive Early Stopping with Label-Level Verifier Margin

**Require:** Claim $c$, evidence set $E$, maximum trace budget $N = 15$, minimum valid traces $n_{min}$, margin threshold $\tau_{margin}$, single-label threshold $n_{single} = 5$
**Ensure:** Final verdict $\hat{y}$

Construct prompt $p$ from claim $c$ and evidence set $E$
Initialize valid trace set $\mathcal{T}_{valid} \leftarrow \emptyset$
**for** $i = 1$ to $N$ **do**
    Generate reasoning trace $t_i$ using prompt $p$
    Parse justification $j_i$ and label $y_i$ from $t_i$
    **if** $y_i \in \{\text{Supports}, \text{Refutes}, \text{Conflicting}\}$ **then**
        Compute verifier score $s_i = V(c, y_i, j_i)$
        Add $(t_i, j_i, y_i, s_i)$ to $\mathcal{T}_{valid}$
    **end if**
    **if** $|\mathcal{T}_{valid}| \geq n_{min}$ **then**
        Compute $b_y = \max_{i:y_i=y} s_i$ for each observed label $y$
        **if** only one label has appeared **then**
            **if** $|\mathcal{T}_{valid}| \geq n_{single}$ **then**
                **break**
            **end if**
        **else**
            Let $y_{(1)}$ and $y_{(2)}$ be the labels with the highest and second-highest $b_y$
            Compute label-level margin $m = b_{y_{(1)}} - b_{y_{(2)}}$
            **if** $m \geq \tau_{margin}$ **then**
                **break**
            **end if**
        **end if**
    **end if**
**end for**
**if** $\mathcal{T}_{valid} = \emptyset$ **then**
    **return** missing-label prediction
**end if**
Select $i^\star = \arg\max_i s_i$ over $\mathcal{T}_{valid}$
**return** $\hat{y} = y_{i^\star}$

To avoid stopping too early, the rule is applied only after a minimum number of valid traces has been verified. When at least two distinct labels have appeared, inference stops if

$$|\mathcal{T}_{valid}| \geq n_{min}$$

and

$$m \geq \tau_{margin}.$$

Here, $\tau_{margin}$ controls how large the verifier-score gap must be before the current prediction is considered stable enough to stop.

A separate condition is used when all valid traces observed so far predict the same label. In that case, no margin can be computed because there is no competing label. The system

therefore stops only after the same label has appeared in a sufficient number of valid traces. In this implementation, the single-label threshold is set to $n_{single} = 5$, which prevents stopping after only one or two agreeing traces.

If margin confidence is reached or if the maximum budget is exhausted, the final prediction is selected using the same verifier-guided Best-of-N rule as before: the system returns the label of the highest-scoring verified trace.

Figure 3.3 provides a visual overview of the proposed method, while Algorithm 2 formalizes the procedure. The examples in Appendix C further illustrate how the Adaptive Stopping margin behaves for unambiguous and ambiguous claims.

*This method is referred to as Adaptive Stopping in Chapter 4 and 5 for brevity.*

## 3.5. Surrogate-Guided Selective Verification

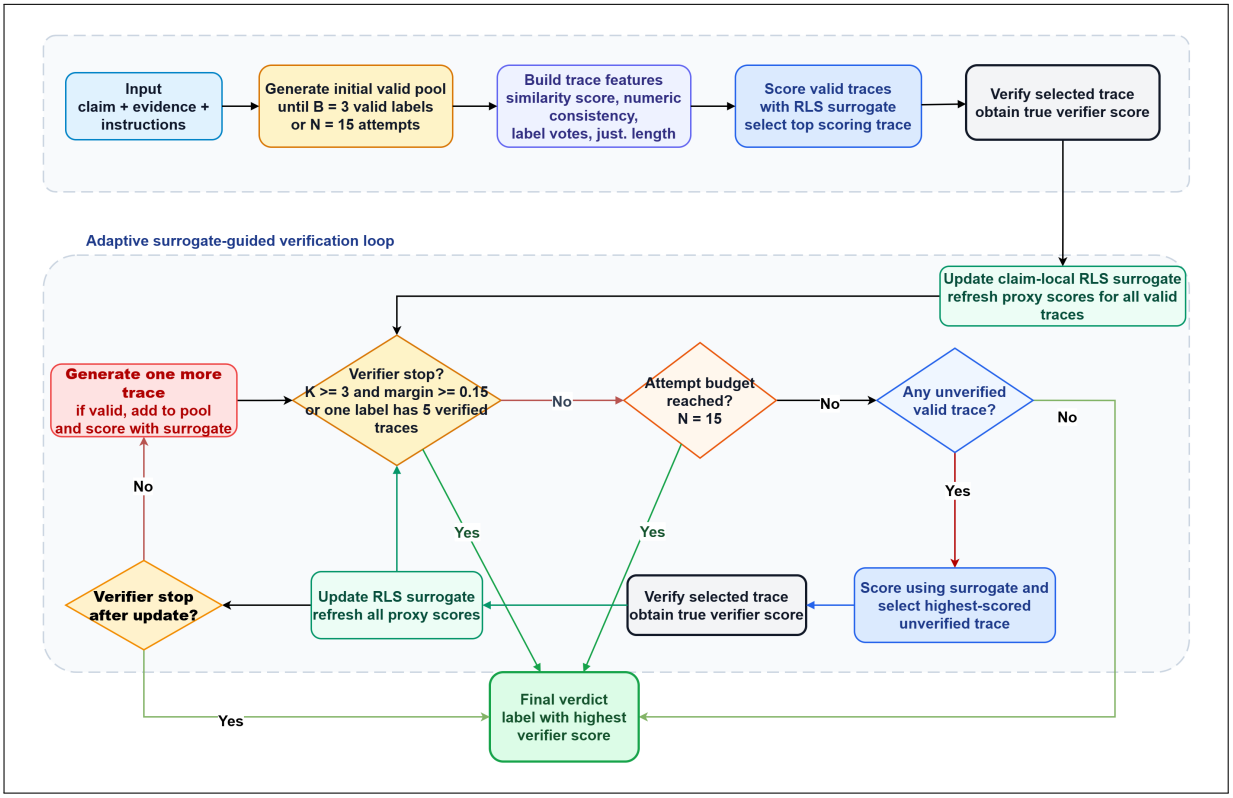



**Figure 3.4:** Overview of Surrogate-Guided Selective Verification, where an online surrogate ranks generated traces before verifier calls are allocated.

The previous section introduced Verifier-Guided Adaptive Stopping as a way to reduce computation by ending trace generation once verified traces show a clear label preference. However, it still sends every valid trace generated to the verifier for scoring. This leaves a second source of inefficiency - not every generated trace is equally useful to verify. Some traces may repeat earlier reasoning, provide weak evidence, or support labels that are unlikely to change the final decision. If verifier calls are expensive, the system should not only decide *when* to stop,

but also *which* traces are worth verifying first.

Surrogate-Guided Selective Verification asks whether verifier calls can also be allocated more selectively. Instead of verifying traces in generation order, the method treats each valid trace as a candidate with unknown utility. In this setting, the utility of a trace is its verifier score, since high-scoring traces are the ones most likely to determine the final verdict. The goal is therefore to identify which traces are likely to be useful before spending a verifier call on them.

To do this, the method uses a lightweight per-claim surrogate model. For each claim, the surrogate estimates the verifier score that an unverified trace is likely to receive. The highest-scoring estimated trace is verified first, the observed verifier score is treated as feedback, and the surrogate is updated online. This creates a sequential selection process where verifier feedback is used not only as a confidence signal for stopping, but also as a reward signal for deciding which trace should be verified next.

This setting is closely related to disposable bandits [60]. Each valid generated trace can be viewed as a disposable arm whose reward is unknown until the trace is verified. Pulling an arm corresponds to sending that trace to the actual verifier and observing its verifier score. Once a trace has been verified, its reward is known and the trace is removed from the unverified candidate pool and is not selected again in later rounds. The surrogate then re-ranks only the remaining unverified traces using the feedback observed so far. In this setting, we treat verification as a sequential allocation problem where each verifier call should be spent on the remaining trace with the highest estimated utility.

It is important to note that the surrogate does not replace the verifier, but rather a mechanism to prioritize verifier calls. Stopping decisions and the final prediction are always based on observed verifier scores, which ensures that the returned verdict depends only on traces that have actually been verified.

### 3.5.1. Per-Claim Linear Surrogate

For each claim, a fresh surrogate $S_c$ is initialized and updated only with verifier feedback from that claim. Each valid trace $t_i$, with justification $j_i$ and parsed label $y_i$, is represented by a feature vector

$$\phi_i \in \mathbb{R}^d.$$

The surrogate predicts the verifier score as

$$\hat{s}_i = S_c(\phi_i) = \mathrm{clip}(w_c^\top \phi_i + b_c, 0, 1),$$

where $\hat{s}_i$ is the estimated utility of verifying trace $t_i$. After a selected trace is sent to the verifier, the observed score $s_i = V(c, y_i, j_i)$ becomes the target used to update the surrogate with Recursive Least Squares (RLS) [63]. Thus, the surrogate adapts during inference using feedback from the current claim.

### 3.5.2. Selective Verification Procedure

The method maintains two sets: a pool of valid generated traces $\mathcal{T}$, and a set of verified traces $\mathcal{V}$. It first generates a small initial pool of valid traces. Each unverified trace is scored by the surrogate, and the trace with the highest estimated utility is sent to the verifier model. The resulting verifier score is then used to update the surrogate.

After each verifier call, the method checks whether the verified traces already provide a clear enough label-level signal to stop. If not, the surrogate re-ranks the remaining unverified traces and selects the next most useful trace to verify. When the generation budget has not yet been exhausted, new traces can also be added to the pool and scored by the updated surrogate.

This gives a verifier-allocation policy: instead of verifying every valid trace, the method prioritizes traces that are estimated to be most useful for the final decision.

### 3.5.3. Stopping and Final Prediction

The stopping rule is the same verifier-margin criterion described in section 3.4, and is computed only over the actual verified traces. If at least two labels have appeared among verified traces, the method stops when the label-level verifier margin exceeds $\tau_{\text{ver}}$ after at least $K$ verifier calls. If all verified traces have the same label, the method stops only after $K_{\text{single}}$ same-label verified traces have been observed.

If the stopping condition is not met, the method continues selecting traces until the generation budget is exhausted or no unverified valid trace remains. The final prediction is selected from verified traces only and for each label, the system uses the best observed verifier score to compute the margin.

Figure 3.4 shows the detailed loop of surrogate-guided selective verification and Algorithm 3 gives the corresponding pseudocode for the proposed method.

## 3.6. Fixed Verifier Budget Trace Selection Analysis

In the previous sections we evaluate end-to-end inference pipeline and how inference cost can be reduced through adaptive generation and surrogate-guided selective verification. However, the surrogate-guided selective verification method combines two decisions: when to stop and which trace to verify next. The fixed verifier budget trace selection analysis isolates the second decision. It asks whether the surrogate-guided ordering itself is useful when the number of verifier calls is fixed in advance.

This analysis is motivated by the core assumption behind Surrogate-Guided Selective Verification - not all generated traces are equally useful to verify. If the surrogate learns a meaningful utility signal from verifier feedback, then the traces it selects early should be more valuable than traces selected by simpler strategies such as generation order or random selection. A fixed verifier budget $k$ directly tests this assumption by asking which $k$ traces should be verified when only a limited number of verifier calls is available.

**Algorithm 3** Surrogate-Guided Selective Verification with Online RLS

**Require:** Claim $c$, evidence $E$, maximum trace attempts $N$, initial valid pool size $B$, minimum verifier updates $K$, verified margin threshold $\tau_{\text{ver}}$, single-label threshold $K_{\text{single}}$
**Ensure:** Final verdict $\hat{y}$

Build prompt $p$ and initialize valid traces $\mathcal{T} \leftarrow \emptyset$, verified traces $\mathcal{V} \leftarrow \emptyset$
Initialize per-claim RLS surrogate $S_c$
Generate traces until $|\mathcal{T}| = B$ valid traces are obtained or $N$ attempts are used; score each valid trace with $S_c$
**if** $\mathcal{T} = \emptyset$ **then**
    **return** missing-label prediction
**end if**
Verify $t^\star = \arg\max_{t_i \in \mathcal{T}} \hat{s}_i$, add it to $\mathcal{V}$, and update $S_c$
Refresh state-dependent features and surrogate scores for all valid traces
**while** true **do**
    Compute verified label scores $b_y = \max_{t_i \in \mathcal{V}: y_i = y} s_i$
    **if** at least two verified labels exist and $|\mathcal{V}| \geq K$ **then**
        Compute verified margin $m_{\text{ver}}$ between the top two $b_y$
        **if** $m_{\text{ver}} \geq \tau_{\text{ver}}$ **then**
            **break**
        **end if**
    **else if** all verified traces have one label and $|\mathcal{V}| \geq K_{\text{single}}$ **then**
        **break**
    **end if**
    **if** $N$ trace attempts have been used **then**
        **break**
    **end if**
    **if** no unverified valid trace exists in $\mathcal{T}$ **then**
        **break**
    **end if**
    Verify $t^\star = \arg\max_{t_i \in \mathcal{T} \setminus \mathcal{V}} \hat{s}_i$
    Add $t^\star$ to $\mathcal{V}$, update $S_c$, and refresh surrogate scores
    Recompute verified label scores $b_y = \max_{t_i \in \mathcal{V}: y_i = y} s_i$
    **if** the verified-margin rule is satisfied **then**
        **break**
    **end if**
    Generate one additional trace if fewer than $N$ attempts have been used; if valid, add it to $\mathcal{T}$ and score it with $S_c$
**end while**
Select $i^\star = \arg\max_{t_i \in \mathcal{V}} s_i$
**return** $\hat{y} = y_{i^\star}$

For each claim, the analysis starts from the valid generated trace pool defined in Section 3.2. A selection strategy orders this pool, and under budget $k$, only the first $k$ traces in that ordering are sent to the verifier. The final label is then computed using the same verifier-score decision rule for every strategy. Thus, the strategies differ only in how they allocate the limited verifier budget, not in the trace representation, verifier, or final prediction rule.

The surrogate-guided condition uses the ordering produced by Surrogate-Guided Selective Verification from Section 3.5. The surrogate is treated as an online selection policy: it selects one trace, observes its verifier score, updates its estimate for the current claim, and then re-ranks the remaining unverified traces. This analysis therefore tests whether the surrogate's estimated utility scores place high-value traces earlier than simpler selection rules.

The following strategies are compared:

- **Generation-order top-$k$.** The first $k$ valid traces are selected in the order they were generated. This corresponds to verifying traces as they appear.
- **Random selection.** A random ordering of the valid traces is generated, and the first $k$ traces are selected for verification. This provides a non-learned allocation baseline.
- **Surrogate-guided selection.** Under verifier budget $k$, the first $k$ traces prioritized by the online surrogate are verified. This tests whether verifier feedback helps select more useful traces than generation-order or random selection.

The analysis is conducted across verifier budgets $k$, where $k$ is constrained to be no larger than the number of available valid generated traces. Comparing these strategies shows whether surrogate-guided selection provides enough benefit to justify its added complexity and overhead.

# 4

# Implementation Details and Experiments

This chapter describes the concrete experimental setup used to evaluate the methods introduced in Chapter 3. It specifies the datasets, models, prompting pipeline, verifier scoring procedure, experimental configurations, runtime tracking, and evaluation metrics. The aim is to make the implementation choices explicit while keeping them separate from the method definitions in the previous chapter.

## 4.1. Dataset

### 4.1.1. QuanTemp

QuanTemp [14] is the primary dataset used throughout the main experiments in this thesis. It is a claim verification dataset consisting of numerical claims collected from real-world fact-checking websites such as PolitiFact, Snopes, and AFP. The claims are particularly suitable for evaluating numerical and temporal reasoning, since they were originally considered claim-worthy by professional fact-checkers. QuanTemp contains claims involving different types of quantitative reasoning, including statistical, temporal, interval, and comparison claims.

This dataset is well suited to adaptive test-time scaling because the correctness of a claim often depends on specific quantities, dates, time spans, rates, or comparison directions. A reasoning trace may therefore fail not only by choosing the wrong label, but also by misreading a number, using the wrong temporal scope, or drawing a conclusion that is not supported by the evidence. This makes QuanTemp a useful setting for studying whether generating multiple reasoning traces and selectively verifying them can improve the final prediction.

The original fact-checking labels are mapped into three veracity classes: *True*, *False*, and *Conflicting*. The full dataset consists of 9,935 training samples, 3,084 validation samples, and 2,495 test samples. The class distribution for each split is shown in Table 4.1.

For the experiments in this thesis, the evaluation is performed on the QuanTemp test split containing 2,495 claims. QuanTemp is used as the main benchmark for the exhaustive, adaptive, surrogate-guided, fixed-budget verifier-allocation, and ablation experiments. Each example

| Dataset split | True | False | Conflicting |
|---|---|---|---|
| Training | 1824 | 5770 | 2341 |
| Validation | 617 | 1795 | 672 |
| Test | 474 | 1423 | 598 |

**Table 4.1:** Distribution of veracity classes in the QuanTemp dataset.

consists of a claim, a gold veracity label, evidence information, and an example identifier used to align generated traces with evaluation outputs. The evidence used during inference is obtained from precomputed BM25 retrieval results: for each claim, candidate evidence passages are semantically re-ranked against the claim, and the top three passages are passed to the generator as the evidence context. The claim text is used as the input statement to be verified, while the selected evidence passages are used to construct the prompt for reasoning trace generation. The gold label is used only for evaluation and is not provided to the generator or verifier during inference.

The label distribution of the test split is imbalanced, with false claims forming the largest class. This makes macro-averaged metrics important in addition to overall accuracy, since accuracy alone may overstate performance on the majority class. The experiments therefore report both predictive quality and inference-time efficiency metrics, allowing Exhaustive BoN, Adaptive Stopping, and Surrogate-Guided Selective Verification to be compared under matched conditions within each dataset.

### 4.1.2. ClaimDecomp

ClaimDecomp [8] is used as a second evaluation dataset to test the proposed adaptive test-time scaling methods beyond numerical claim verification. The dataset focuses on complex political claims, which are broken down into yes-no sub-questions. These sub-questions cover both explicit propositions in the original claim and implicit contextual facets that may affect the final veracity judgement. This makes ClaimDecomp useful for evaluating whether adaptive trace generation and selective verification remain effective in a decompositional fact-checking setting, where the model must reason over multiple claim components rather than verify a single surface-level statement.

The test split used in our experiments contains 200 claims, with a label distribution of 60 true (30.0%), 91 false (45.5%), and 49 conflicting (24.5%), indicating a moderate class imbalance skewed toward false claims.

## 4.2. Models

### 4.2.1. Generator

Reasoning traces are generated using Llama 3.2 3B [64], a 3-billion-parameter instruction-tuned language model. The model is chosen because it is small enough to run locally through

an Ollama inference backend, making repeated multi-trace generation computationally feasible under the available hardware constraints. Each trace is generated with a maximum output length of 512 tokens and top-$p$ sampling with $p = 1.0$. The decoding temperature is varied across traces according to a balanced schedule ranging from 0.30 to 0.70 in steps of 0.05, as described in Section 4.4.

### 4.2.2. Verifier

The verifier model is adopted from [29]. It is based on Llama 3.2 3B fine-tuned with Low-Rank Adaptation (LoRA) as a binary consistency classifier. The model takes a concatenation of the claim, predicted verdict, and generated justification as input and produces a scalar confidence score via the sigmoid function. LoRA is applied with rank $r = 8$ and scaling parameter $\alpha = 16$. During inference, verifier inputs are padded or truncated to a fixed maximum length before scoring.

Unless otherwise stated, this LoRA fine-tuned Llama 3.2 3B verifier is used in the main experiments. A separate verifier ablation replaces this model with DeepSeek-R1-Distill-Qwen-7B [65], served through Ollama and referred to as DeepSeek-R1:7B throughout this thesis, while keeping the generator fixed as Llama 3.2 3B. DeepSeek-R1-Distill-Qwen-7B is chosen because it is a small, open-weight reasoning model, enabling a comparison between the task-specific LoRA verifier and a prompt-based general reasoning model of similar size.

This ablation tests whether the proposed optimization algorithms generalize across verifier models and whether the efficiency gains remain consistent when the verifier formulation changes. Unlike the LoRA verifier, DeepSeek-R1:7B is used as a prompt-based verifier rather than as a task-specific classifier. For each claim, predicted verdict, and generated justification, the system sends a structured scoring prompt to DeepSeek-R1:7B and asks it to return a JSON score in $[0, 1]$. The resulting score is therefore produced through instruction following and score extraction, not through a fine-tuned verification head. The full prompt is provided in Appendix A.5.

### 4.2.3. Embedding Model

Semantic similarity computations use the `paraphrase-MiniLM-L6-v2` sentence-transformer model [66]. In the QuanTemp pipeline, this model is used to re-rank BM25 candidate documents by cosine similarity to the claim. In both datasets, it is also used during prompt construction to select semantically similar few-shot examples from the annotated pool.

## 4.3. Evidence Retrieval and Re-ranking

For QuanTemp, the experiments use precomputed BM25 retrieval results as the initial evidence source. For each claim, BM25 provides a ranked list of candidate evidence passages from the QuanTemp evidence corpus. Because BM25 mainly captures lexical overlap, these candidates are re-ranked using semantic similarity before being passed to the generator. For

ClaimDecomp, the available decompositional evidence fields are used directly as the evidence context.

For QuanTemp, each claim and candidate passage is encoded with the sentence-transformer embedding model, and cosine similarity is used to rank the candidates. The top three semantically ranked passages are selected as the evidence context for prompt construction. If too few passages satisfy the similarity threshold, the threshold is relaxed and the highest-ranked candidates are used instead. This ensures that every QuanTemp claim is evaluated with an evidence context while still prioritizing passages that are semantically close to the claim.

## 4.4. Prompt Construction and Reasoning Trace Generation

After the evidence passages are selected, a prompt is constructed for each claim. The prompt contains the claim, the retrieved evidence, and up to four semantically similar few-shot examples. The few-shot examples are selected by embedding the current claim and comparing it with claims from a held-out set of annotated examples. The nearest examples are included to demonstrate the expected reasoning format and output structure.

Each prompt instructs the generator to reason over the evidence and produce a single justification for the whole claim, followed by one final veracity label. The model is explicitly instructed to output the final answer using the fields `[Justification]:` and `[Label]:`. The label is constrained to one of the three supported classes:

$$\{\text{Supports}, \text{Refutes}, \text{Conflicting}\}.$$

Reasoning traces are generated using an instruction-tuned language model served through a local inference backend. For each claim, multiple traces are sampled by varying the decoding temperature. The temperature grid ranges from 0.30 to 0.70 with a step size of 0.05. A balanced temperature schedule is used so that traces are sampled across different temperature values rather than repeatedly using a single decoding setting. This encourages diversity among the generated reasoning paths while keeping the maximum trace budget fixed.

In the main exhaustive setting, the generator produces up to 15 reasoning traces for each claim. In the adaptive and surrogate-guided settings, the same maximum budget is used, but generation may terminate earlier when the verified-margin stopping criterion is satisfied. Additional budget analyses use larger candidate pools where available. For every generated trace, the system records the generated text, decoding temperature, generation latency, and token counts. These quantities are later used to compare the computational cost of the different test-time scaling strategies.

## 4.5. Verifier Trace Scoring

The verifier model is used to assign a confidence score to each valid generated trace. Its input consists of the original claim, the predicted verdict, and the generated justification:

$$x_i = [c; y_i; j_i],$$

where $c$ is the claim, $y_i$ is the predicted label, and $j_i$ is the generated justification for trace $t_i$.

In the main experiments, the verifier is implemented as a fine-tuned transformer-based classifier with LoRA adaptation. It produces a scalar logit for each claim-verdict-justification triple. This logit is converted into a confidence score using the sigmoid function:

$$s_i = \sigma(f_\theta(c, y_i, j_i)).$$

The score $s_i$ represents the verifier's confidence that the generated justification supports the proposed verdict for the claim.

In the exhaustive and Adaptive Stopping settings, every valid trace is scored by the verifier until the trace budget is exhausted or the Adaptive Stopping rule is triggered. In the Surrogate-Guided Selective Verification setting, only selected traces are scored by the verifier, while the remaining valid traces may receive estimated scores from the surrogate model. The final prediction is always selected from verifier-scored traces.

## 4.6. Experimental Configurations

Within each main dataset comparison, experiments use the same evidence handling, prompt construction, generator, and verifier model. The variants differ in how reasoning traces are generated, verified, selected, and stopped during inference. This setup keeps the main comparisons focused on the test-time scaling strategy, while the ablation studies intentionally vary the verifier model and trace budgets.

### 4.6.1. Research Question Mapping

Table 4.2 summarizes how the experimental configurations answer the research questions introduced in Chapter 1.

### 4.6.2. Exhaustive Test-Time Scaling

The exhaustive baseline uses a fixed trace budget of 15 reasoning traces per claim. Adaptive Stopping is disabled in this setting. For each claim, the generator attempts to produce all 15 traces, and every trace with a valid parsed label is passed to the verifier. The final prediction is selected from the valid verified traces using the highest verifier score.

This configuration provides the full-budget reference point for the experiments. It is expected to have the highest generator and verifier usage, since it does not attempt to reduce computation

**Table 4.2:** Mapping between research questions and experimental configurations.

| Question | Main comparison | Purpose |
|---|---|---|
| RQ1 | Exhaustive BoN ($N = 15$) vs. Adaptive Stopping | Tests whether margin-based stopping reduces trace generation while preserving verification performance. |
| RQ2 | Adaptive Stopping vs. Surrogate-Guided Selective Verification | Tests whether surrogate-guided trace selection can further reduce verifier usage while maintaining performance. |
| RQ3 | Generation-order top-$k$, random selection, and surrogate-guided selection | Tests whether the surrogate ordering allocates a fixed verifier budget more effectively than simpler selection rules. |
| Ablations | Larger trace budgets and DeepSeek-R1:7B verifier | Tests whether the trade-off changes when the candidate pool grows or when the verifier model is changed. |

during inference.

### 4.6.3. Verifier-Guided Adaptive Stopping

The Adaptive Stopping configuration uses the same maximum budget of 15 traces per claim, but allows inference to terminate earlier. After each valid trace is generated and verified, the system computes a label-level verifier-score margin over the valid traces generated so far.

The verifier margin is used as a confidence signal: if the highest label-level verifier score is sufficiently separated from the second-highest label-level score, the current best label is treated as stable enough to stop. The margin threshold is set to $0.15$, which gives the method a moderate stopping criterion: it avoids continuing when the verifier already shows a clear preference, but does not stop on very small score differences.

Early stopping is enabled only after at least three valid traces have been obtained. This minimum prevents the method from stopping based on one or two stochastic samples, which may not represent the range of possible reasoning paths for the claim. If all valid traces observed so far have the same label, stopping is delayed until five same-label valid traces have accumulated. This handles cases where there is no competing label yet; repeated same-label traces provide stronger evidence of stability than a small initial sample. If neither condition is satisfied, trace generation continues until either the stopping rule is triggered or the maximum budget of 15 traces is reached.

### 4.6.4. Surrogate-Guided Selective Verification

The Surrogate-Guided Selective Verification configuration also uses a maximum budget of 15 trace attempts per claim in the main setting, but it does not send every valid trace to the verifier. Instead, it uses an online linear surrogate to estimate verifier scores for unverified traces and prioritize which trace should be verified next. The surrogate is initialized separately for each claim and updated only with verifier feedback obtained from that claim's traces.

For each claim, the method first generates a bootstrap pool of three valid traces. This gives the surrogate several candidates to rank before the first verifier call. The highest-scoring trace in this pool is sent to the verifier, and the observed verifier score is used to update the RLS surrogate. After this update, all valid traces are re-scored, and subsequent verifier calls are allocated to the currently highest-scoring unverified trace.

The procedure combines selective verification with early stopping. Generation and verification continue until the maximum trace budget is reached, the verified-score margin is large enough, or no unverified valid trace remains. A new trace is generated only if the verified-margin stopping rule is not satisfied. The final prediction is selected only from verifier-scored traces, ensuring that the final verdict remains grounded in actual verifier outputs.

**Table 4.3:** Main configuration for Surrogate-Guided Selective Verification.

| Parameter | Value | Role |
|---|---|---|
| Maximum trace attempts $N$ | 15 | Upper bound on generated traces per claim |
| Bootstrap pool $B$ | 3 valid traces | Candidate pool before the first verifier call |
| Minimum verifier updates $K$ | 3 | Minimum feedback before margin-based stopping |
| Verified margin $\tau_{\text{ver}}$ | 0.15 | Stopping threshold for observed verifier scores |
| Single-label threshold | 5 traces | Minimum evidence when only one label appears |

Table 4.3 lists the main surrogate configuration. The verified-margin threshold matches the Adaptive Stopping threshold because both methods use observed verifier scores for stopping. The surrogate is used only to rank unverified traces for verification.

### 4.6.5. Surrogate Features and Online Update

The surrogate uses a seven-dimensional feature vector. The features are: claim-justification similarity, evidence-justification similarity, claim numeric coverage, evidence numeric grounding, label vote ratio, same-label verifier score, and normalized justification length. These features were selected because they are inexpensive to compute, interpretable, and available before calling the verifier on a new trace.

The feature set combines static trace information and dynamic state information. Similarity and numeric-grounding features describe the generated justification before verification. The label vote ratio and same-label verifier score summarize the current state of the trace pool after earlier verifier calls. These history-based features are important because the surrogate must adapt from only a small number of verifier observations for the current claim.

The surrogate is trained using recursive least squares (RLS) [63], an online version of linear least squares. After each verifier call, the surrogate receives the selected trace feature vector and observed verifier score, computes the prediction error, and updates its weights immediately. This makes RLS suitable for the adaptive pipeline, where verifier feedback arrives

sequentially and must be used during the same inference run.

### 4.6.6. Fixed Verifier Budget Trace Selection

For the fixed verifier budget trace selection experiment, candidate traces are generated first and the selection strategies are applied offline to the resulting valid trace pool. For each budget $k$, generation-order top-$k$, random selection, and surrogate-guided selection are evaluated using the same verifier-score decision rule. Random selection is run with a fixed seed to ensure reproducibility.

The fixed verifier budget analysis is evaluated on three generated trace pools. The QuanTemp test setting uses the same $N = 15$ trace pool as the main experiments. The ClaimDecomp test setting uses a larger $N = 50$ trace pool to evaluate selection when more candidate traces are available. A larger-pool QuanTemp validation setting is also included, using a stratified sample of 500 validation claims with up to 50 traces per claim. This keeps the experiment computationally feasible while preserving the validation label distribution.

## 4.7. Ablation Study

Two ablation studies are used to support the main research questions. The first varies the maximum reasoning trace budget. In addition to the main $N = 15$ setting, larger trace budgets of $N = 20$, $N = 30$, and $N = 40$ are evaluated. This tests how the cost-performance trade-off between Exhaustive BoN and Adaptive Stopping changes when the maximum trace budget increases. All other components, including the dataset, evidence retrieval procedure, prompt format, generator, verifier, and verified-margin threshold, are held fixed.

The second ablation changes the verifier model. The main experiments use Llama 3.2 3B as both the generator backbone and the LoRA fine-tuned verifier backbone. In the verifier ablation, the generator remains Llama 3.2 3B, but the verifier is replaced by DeepSeek-R1:7B. This isolates the effect of verifier choice on prediction quality, verifier-call cost, latency, and the effectiveness of adaptive or surrogate-guided allocation.

## 4.8. Runtime and Cost Tracking

For the efficiency evaluation, the pipeline records both operation counts and latency information. For each claim, it records the number of generated trace attempts, the number of valid traces, missing-label traces, and verifier calls. These quantities are used to compare how much computation is required by each experimental variant.

The implementation separately tracks generation latency and verification latency. Generation latency is measured for each call to the generator, while verification latency is measured for each verifier scoring operation. The total processing time per claim is also recorded. This allows the experiments to distinguish between savings obtained from generating fewer traces and savings obtained from making fewer verifier calls.

The pipeline also records token usage for generation, including prompt tokens and completion tokens. These values are useful for estimating the cost of language-model-based generation and for understanding how trace sampling affects inference-time resource usage.

All experiments were run on NVIDIA A100 GPUs. For the energy-cost estimate, the GPU power draw is approximated as $P = 0.25$ kW and the electricity price is set to $q = 0.25$ euros per kilowatt-hour, matching the Netherlands electricity-price assumption used in the experiment configuration. These values are used to obtain a consistent comparative cost estimate across methods, rather than an exact hardware billing measurement.

An estimated GPU energy cost is computed using this configured GPU power draw and electricity price. If $t$ is the runtime in seconds, $P$ is the GPU power draw in kilowatts, and $q$ is the electricity price in euros per kilowatt-hour, the estimated cost is

$$\text{cost}(t) = \frac{t}{3600} \cdot P \cdot q.$$

This estimate is reported separately for generation, verification, and total claim processing time.

## 4.9. Evaluation Metrics

### 4.9.1. Predictive Performance Metrics

Predictive performance is measured by comparing the predicted label for each claim against the corresponding gold label. The reported classification metrics are class-wise F1 scores, Macro-F1, Weighted-F1, and Accuracy.

Macro-F1 is used as the primary metric throughout the results chapter. This choice is motivated by the imbalanced label distributions in both datasets, where false claims constitute the majority class. By assigning equal weight to each class, Macro-F1 prevents performance on majority classes from dominating the overall score and provides a balanced assessment across all labels. Weighted-F1 accounts for class imbalance by weighting each class according to its support, while Accuracy measures the proportion of correctly classified claims regardless of class frequency.

Throughout the experiments, True/False/Conflicting dataset labels are mapped to Supports/Refutes/Conflicting model labels. In the result tables, T-F1, F-F1, and C-F1 correspond to Supports/True, Refutes/False, and Conflicting respectively.

### 4.9.2. Inference Efficiency Metrics

Inference efficiency is measured using operation counts, latency, token usage, and estimated cost. Operation counts include the total number of generated trace attempts, the number of valid traces, the number of missing-label traces, and the number of verifier calls. For Adaptive Stopping and Surrogate-Guided Selective Verification, these metrics indicate how much computation is saved relative to Exhaustive BoN.

Latency is reported at both the claim level and the trace level. Generation latency is measured per claim and per generated trace, while verification latency is measured per claim and per verifier call. Total claim processing time is also reported. Finally, the estimated GPU energy cost is reported to compare the monetary efficiency of the different test-time scaling strategies.

Together, these metrics allow the experiments to evaluate the central trade-off in this thesis: whether adaptive and surrogate-guided test-time scaling can reduce inference cost while preserving claim verification performance.

# 5

# Results

## 5.1. Overview

This chapter presents the experimental results for verifier-guided test-time scaling on the QuanTemp and ClaimDecomp datasets. It first motivates the use of Exhaustive Best-of-N (BoN) decoding as the main baseline by comparing it with top-1 decoding and self-consistency. The remaining sections then answer the three research questions by evaluating whether Verifier-Guided Adaptive Stopping and Surrogate-Guided Selective Verification can reduce inference cost while maintaining verification performance across both datasets. The chapter also evaluates surrogate-guided selection under fixed verifier budgets, where the generated trace pool is held fixed and only the allocation of verifier calls is varied. Unless stated otherwise, QuanTemp results use Llama 3.2-3B as the generator and a LoRA fine-tuned Llama 3.2-3B model as the verifier. The ablation studies separately examine two settings: replacing the LoRA verifier with DeepSeek-R1:7B, and increasing the maximum trace budget.

Macro-F1 is used as the primary performance metric. Accuracy is reported as a secondary metric for interpretability, while Weighted-F1 and class-wise F1 scores are included in the tables for completeness.

## 5.2. Establishing Exhaustive Best-of-N as the Strong Baseline

Before evaluating adaptive and surrogate-guided variants, this section compares three decoding strategies: top-1 decoding, self-consistency, and Exhaustive BoN. The purpose of this comparison is to establish whether Exhaustive BoN provides a strong enough performance baseline to justify using it as the reference point for the rest of the chapter.

### 5.2.1. Setup

Top-1 decoding generates a single reasoning trace for each claim and uses the parsed label from that trace as the prediction. Self-consistency generates multiple traces and predicts the label by majority vote over the parsed trace labels. Exhaustive BoN also generates multiple traces, but it scores every valid trace with the verifier and selects the label from the trace with the highest verifier score.

All three decoding strategies are evaluated on the same dataset splits using the same generator, prompting format, label parser, and evaluation metrics. This keeps the comparison focused on the decoding and selection strategy.

### 5.2.2. Baseline Performance Comparison

**Table 5.1:** Baseline comparison of top-1, self-consistency, and Exhaustive BoN.

| Dataset | Method | T-F1 | F-F1 | C-F1 | M-F1 | W-F1 | Acc. |
|---|---|---|---|---|---|---|---|
| QuanTemp | Top-1 decoding | 33.03 | 50.23 | 36.54 | 39.93 | 43.68 | 41.88 |
| | Self-consistency | 34.34 | 50.35 | 38.20 | 40.96 | 44.40 | 42.73 |
| | Exhaustive BoN | **40.05** | **68.20** | **46.23** | **51.49** | **57.58** | **55.71** |
| ClaimDecomp | Top-1 decoding | 30.61 | 49.06 | 40.56 | 40.08 | 41.44 | 41.50 |
| | Self-consistency | 23.16 | **54.55** | 35.76 | 37.82 | 40.53 | 40.00 |
| | Exhaustive BoN | **35.42** | 44.96 | **43.43** | **41.27** | **41.72** | **42.00** |

Table 5.1 shows that, on QuanTemp, Exhaustive BoN is clearly stronger than both single-sample decoding and self-consistency. Compared with top-1 decoding, Exhaustive BoN improves Macro-F1 from 39.93 to 51.49 and accuracy from 41.88 to 55.71. Compared with self-consistency, Exhaustive BoN improves Macro-F1 by 10.53 percentage points and accuracy by 12.98 percentage points. On ClaimDecomp, Exhaustive BoN reaches the highest Macro-F1 and accuracy among the three decoding strategies, with a score of 41.27 and 42.00 respectively.

**Key Insights**

*Table 5.1 shows that **Exhaustive BoN is the strongest overall decoding baseline**. On QuanTemp, it gives the clearest improvement, increasing Macro-F1 from 39.93 for top-1 decoding and 40.96 for self-consistency to **51.49**. On ClaimDecomp, the gains are smaller but the same overall pattern holds: Exhaustive BoN obtains the best Macro-F1 score of **41.27**. Therefore, Table 5.1 supports using Exhaustive BoN as the high-compute reference baseline for the remaining experiments.*

## 5.3. Verifier-Guided Adaptive Stopping Reduces Inference Cost While Preserving Performance

This section evaluates whether verifier feedback can stop reasoning trace generation early without substantially reducing verification performance. Verifier-Guided Adaptive Stopping is compared with Exhaustive BoN under matched experimental settings. The analysis examines both verification performance and reductions in trace attempts, verifier calls, generation tokens, latency, runtime, and estimated cost. *For brevity, Verifier-Guided Adaptive Stopping is referred to as Adaptive Stopping in the result tables.*

**Table 5.2:** Verification performance comparison between Exhaustive BoN and Adaptive Stopping ($N = 15$).

| Dataset | Method | T-F1 | F-F1 | C-F1 | M-F1 | W-F1 | Acc. |
|---|---|---|---|---|---|---|---|
| QuanTemp | Exhaustive BoN (baseline) | **40.05** | **68.20** | **46.23** | **51.49** | **57.58** | **55.71** |
| | Adaptive Stopping | 38.86 | 65.21 | 44.15 | 49.41 | 55.16 | 53.27 |
| ClaimDecomp | Exhaustive BoN (baseline) | **35.42** | 44.96 | **43.43** | 41.27 | 41.72 | 42.00 |
| | Adaptive Stopping | 35.05 | **51.75** | 38.75 | **41.85** | **43.55** | **42.50** |

### 5.3.1. Performance Under Verifier-Guided Adaptive Stopping

Table 5.2 shows that, on QuanTemp, Exhaustive BoN obtains the strongest observed verification performance, with a Macro-F1 of 51.49 and an accuracy of 55.71. Adaptive Stopping reaches 49.41 Macro-F1 and 53.27 accuracy, corresponding to decreases of 2.08 and 2.44 percentage points, respectively. On ClaimDecomp, Adaptive Stopping slightly outperforms Exhaustive BoN, with Macro-F1 increasing by 0.58 percentage points, from 41.27 to 41.85, and accuracy increasing by 0.50 percentage points, from 42.00 to 42.50. Because the methods use independently generated stochastic trace sets, these differences represent run-level outcomes rather than claim-matched comparisons. Overall, Adaptive Stopping maintains comparable performance while using substantially less computation, as shown in Table 5.3.

### 5.3.2. Generation and Verification Savings

**Table 5.3:** Generator and verifier call savings from Adaptive Stopping ($N = 15$).

| Method | Avg traces /claim | Trace attempts | Missing-label traces | Verifier calls | Total operations | Operations reduction |
|---|---|---|---|---|---|---|
| **QuanTemp** | | | | | | |
| Exhaustive BoN (baseline) | 15.0 | 37,425 | 327 | 37,098 | 74,523 | – |
| Adaptive Stopping | **9.2** | **22,915** | 196 | **22,719** | **45,634** | $-38.8\%$ |
| **ClaimDecomp** | | | | | | |
| Exhaustive BoN (baseline) | 15.0 | 3,000 | 16 | 2,984 | 5,984 | – |
| Adaptive Stopping | **7.2** | **1,446** | 5 | **1,441** | **2,887** | $-51.8\%$ |

**Table 5.4:** RQ1 generation-token savings from Adaptive Stopping ($N = 15$).

| Method | Prompt tokens | Completion tokens | Total gen. tokens | Token reduction |
|---|---|---|---|---|
| **QuanTemp** | | | | |
| Exhaustive BoN (baseline) | 66,057,300 | 6,139,215 | 72,196,515 | – |
| Adaptive Stopping | **40,357,672** | **3,736,075** | **44,093,747** | $-38.9\%$ |
| **ClaimDecomp** | | | | |
| Exhaustive BoN (baseline) | 5,514,480 | 447,030 | 5,961,510 | – |
| Adaptive Stopping | **2,652,330** | **213,707** | **2,866,037** | $-51.9\%$ |

Tables 5.3 and 5.4 show that the performance trade-off is accompanied by a substantial reduction in computation. On QuanTemp, Adaptive Stopping lowers the average number of generated traces from 15.0 to 9.2 per claim, reducing trace attempts by 14,510 and verifier calls by 14,379. In total, this reduces the total operations (generation + verification) by 28,889, corresponding to a 38.8% reduction, while generation tokens also decrease by 38.9%. On ClaimDecomp, the savings are even larger. Total operations decrease by 3,097 (51.8%), and

generation tokens decrease by 51.9%. These savings are driven by early termination of trace generation, which in turn reduces the number of corresponding verifier calls.

The claim-level analysis in Table B.1 provides further context for these aggregate savings. On QuanTemp, Adaptive Stopping terminates before reaching the 15-trace cap for 1,385 of 2,495 claims (55.5%). The table further shows that 1,299 claims (52.1%) use no more than five valid traces, while another 67 use between six and ten. On ClaimDecomp, Adaptive Stopping terminates early for 142 of 200 claims (71%). These results confirm that the overall reductions in operations and tokens arise because a large proportion of claims do not require the complete trace budget.

### 5.3.3. Latency, Runtime, and Cost Savings

**Table 5.5:** Latency, estimated total processing runtime, and estimated cost comparison ($N = 15$). Runtime is computed from summed per-claim processing latency and corresponds to the estimated cost column.

| Dataset | Method | Gen./claim (s) | Ver./claim (s) | Total/claim (s) | Runtime (h) | Cost (€) |
|---|---|---|---|---|---|---|
| QuanTemp | Exhaustive BoN (baseline) | 16.37 | 12.07 | 28.44 | 19.71 | 1.232 |
| | Adaptive Stopping | **9.30** | **7.20** | **16.50** | **11.43** | **0.715** |
| ClaimDecomp | Exhaustive BoN (baseline) | 13.62 | 11.76 | 25.39 | 1.41 | 0.088 |
| | Adaptive Stopping | **7.49** | **6.80** | **14.29** | **0.79** | **0.050** |

Table 5.5 shows that the operation savings translate into lower end-to-end latency, estimated processing runtime, and estimated GPU energy cost. On QuanTemp, total latency falls from 28.44 to 16.50 seconds per claim (42.0%), while estimated processing runtime falls from 19.71 to 11.43 hours, a 42.0% reduction. On ClaimDecomp, total latency falls from 25.39 to 14.29 seconds per claim, while estimated processing runtime falls from 1.41 to 0.79 hours, a 43.7% reduction in inference runtime.

**Key Insights**

1. *Large efficiency gains with only a small performance drop on QuanTemp. Adaptive Stopping reduces total operations by **38.8%**, generation tokens by **38.9%**, and estimated runtime by **42.0%**. This comes with a modest decrease of **2.08 percentage points** in Macro-F1 and **2.44 percentage points** in accuracy relative to Exhaustive BoN.*
2. *Even larger savings on ClaimDecomp without a performance loss in this run. Adaptive Stopping reduces total operations by **51.8%**, generation tokens by **51.9%**, and estimated runtime by **43.7%**. At the same time, Macro-F1 increases by **0.58 percentage points** and accuracy by **0.50 percentage points** compared with Exhaustive BoN.*
3. *The efficiency gain is driven by early trace-generation stopping. By stopping before the full trace budget is used, Adaptive Stopping avoids unnecessary generation calls; the reduction in verifier calls follows because fewer generated traces need to be verified.*

## 5.4. Surrogate Guidance Further Reduces Verifier Usage

This section examines whether Surrogate-Guided Selective Verification can reduce verifier-side cost beyond the savings already achieved by Adaptive Stopping. Unlike Adaptive Stopping, which saves computation mainly by stopping trace generation early, the surrogate-guided method targets the verification stage directly by estimating which generated traces are most worth sending to the verifier.

### 5.4.1. Verifier-Call Savings Under Surrogate Guidance

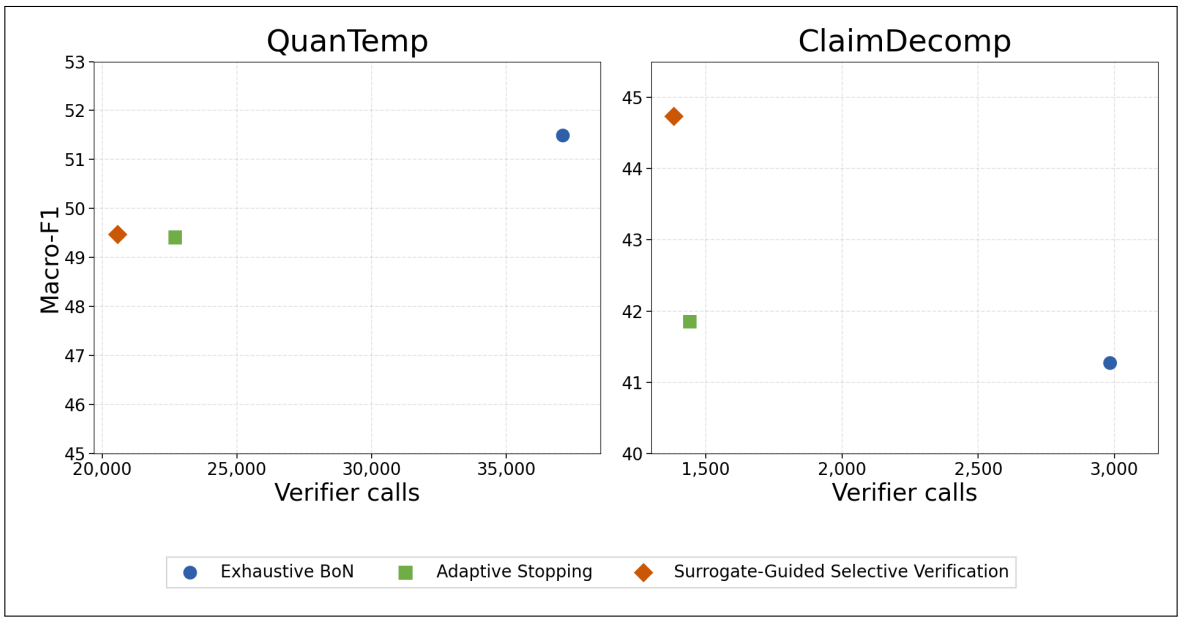



**Figure 5.1:** Verifier-call and Macro-F1 trade-off across methods ($N = 15$).

Figure 5.1 compares Exhaustive BoN, Adaptive Stopping, and Surrogate-Guided Selective Verification in terms of verifier calls and Macro-F1. The figure shows whether each method reduces verifier usage while staying close to the performance of the high-compute Exhaustive BoN baseline. On QuanTemp, Surrogate-Guided Selective Verification uses the fewest verifier calls, with 20,582 calls compared with 22,719 for Adaptive Stopping and 37,098 for Exhaustive BoN, while remaining in the same run-level Macro-F1 range. On ClaimDecomp, the surrogate-guided run again uses the fewest verifier calls, with 1,383 calls compared with 1,441 for Adaptive Stopping and 2,984 for Exhaustive BoN, while reaching 44.73 Macro-F1 in this run.

### 5.4.2. Runtime and Cost Effects of Surrogate Guidance

**Table 5.6:** Trace attempts and verifier-call savings from surrogate-guided verification ($N = 15$).

| Method | Trace attempts | Missing-label traces | Total verifier calls | Verifier calls/ claim | Savings vs. Exh. BoN | Savings vs. Adaptive |
|---|---|---|---|---|---|---|
| **QuanTemp** | | | | | | |
| Exhaustive BoN (baseline) | 37,425 | 327 | 37,098 | 14.87 | – | – |
| Adaptive Stopping | **22,915** | 196 | 22,719 | 9.11 | 38.8% | – |
| Surrogate-Guided Selective Verification | 24,255 | 158 | **20,582** | **8.25** | **44.5%** | **9.4%** |
| **ClaimDecomp** | | | | | | |
| Exhaustive BoN (baseline) | 3,000 | 16 | 2,984 | 14.92 | – | – |
| Adaptive Stopping | **1,446** | 5 | 1,441 | 7.20 | 51.7% | – |
| Surrogate-Guided Selective Verification | 1,645 | 5 | **1,383** | **6.92** | **53.7%** | **4.0%** |

Table 5.6 shows the verifier-call savings obtained by adding surrogate-guided selection. On QuanTemp, Surrogate-Guided Selective Verification makes 20,582 verifier calls, compared with 22,719 for Adaptive Stopping and 37,098 for Exhaustive BoN baseline. This corresponds to a 44.5% reduction relative to the baseline and an additional 9.4% reduction relative to Adaptive Stopping, despite requiring more trace attempts than Adaptive Stopping. The higher trace-attempt count comes from the surrogate bootstrap phase described in Chapter 4, where extra candidate traces are generated for ranking but only selected traces are sent to the verifier. The same pattern appears on ClaimDecomp: the surrogate-guided method uses 1,383 verifier calls, saving 53.7% relative to Exhaustive BoN and a further 4.0% relative to Adaptive Stopping.

Table 5.7 shows that verifier-call savings do not always translate directly into lower total runtime or total cost. On QuanTemp, Surrogate-Guided Selective Verification reduces verifier runtime relative to Adaptive Stopping (4.52 vs. 4.99 hours), but increases generation runtime (7.85 vs. 6.44 hours), so its total runtime remains higher (12.41 vs. 11.43 hours). On ClaimDecomp, the same pattern appears at a smaller scale: the surrogate lowers verifier runtime from 0.38 to 0.30 hours, but increases generation runtime from 0.42 to 0.51 hours. Thus, the surrogate is best interpreted as reducing verifier-side cost, while Adaptive Stopping remains the lowest total-runtime and lowest total-cost configuration in these runs.

**Table 5.7:** Estimated generation-and-verification runtime breakdown and cost comparison ($N = 15$). Runtime is computed from summed per-claim generation and verification latency.

| Method | Gen. runtime (h) | Ver. runtime (h) | Total runtime (h) | Verifier cost (€) | Total cost (€) |
|---|---|---|---|---|---|
| **QuanTemp** | | | | | |
| Exhaustive BoN (baseline) | 11.34 | 8.37 | 19.71 | 0.523 | 1.232 |
| Adaptive Stopping | **6.44** | 4.99 | **11.43** | 0.312 | **0.715** |
| Surrogate-Guided Selective Verification | 7.85 | **4.52** | 12.41 | **0.282** | 0.775 |
| **ClaimDecomp** | | | | | |
| Exhaustive BoN (baseline) | 0.76 | 0.65 | 1.41 | 0.041 | 0.088 |
| Adaptive Stopping | **0.42** | 0.38 | **0.79** | 0.024 | **0.050** |
| Surrogate-Guided Selective Verification | 0.51 | **0.30** | 0.81 | **0.019** | 0.051 |

The complete wall-clock runtime, latency, and cost breakdown is reported in Appendix B.2. Since the surrogate-side processing was not timed as a separate component in the saved metrics, Table B.3 reports a conservative upper bound using the residual non-generation and non-verifier runtime of at most 0.58 hours (€0.036) on QuanTemp and 0.05 hours (€0.003) on ClaimDecomp. The actual RLS prediction/update cost is smaller because this residual also includes evidence retrieval, prompt assembly, surrogate feature construction, logging, and file I/O.

Supplementary analyses, including trace-bucket performance, wall-clock runtime comparisons, and surrogate-overhead diagnostics, are provided in Appendix Sections B.1–B.2.

**Key Insights**

1. *Surrogate guidance reduces verifier usage while maintaining comparable verification performance. On QuanTemp, it reduces verifier calls by **44.5%** relative to Exhaustive BoN while remaining in the same run-level Macro-F1 range. On ClaimDecomp, it reduces verifier calls by **53.7%** relative to Exhaustive BoN and reaches **44.73** Macro-F1, compared with **41.27** for the baseline.*
2. *The verifier-call savings also translate into lower verifier-side runtime. Compared with Adaptive Stopping, surrogate guidance reduces verifier runtime from **4.99** to **4.52** hours on QuanTemp and from **0.38** to **0.30** hours on ClaimDecomp.*
3. *The benefit is mainly verifier-side rather than end-to-end. Surrogate guidance lowers verifier calls, verifier runtime, and verifier cost, but Adaptive Stopping remains the lowest total-runtime and lowest total-cost configuration because it generates fewer traces overall.*

## 5.5. Surrogate-Guided Selection Prioritizes Better Traces Under Fixed Verifier Budgets

This section evaluates surrogate-guided selection under a fixed verifier budget. Unlike the previous experiments, which compare complete inference pipelines, this analysis holds the

generated trace pool fixed and isolates the verifier-call allocation decision. For a budget of $k$ verifier calls, the question is whether the surrogate can select more useful traces for verification than generation-order or random selection. The evaluation is performed retrospectively on three trace pools: QuanTemp test with $N = 15$, ClaimDecomp test with $N = 50$, and QuanTemp validation with $N = 50$. This allows the analysis to assess surrogate-guided selection under the primary experimental setting and determine whether its benefits persist across datasets and when the number of available candidate traces increases.

### 5.5.1. Fixed-Budget Selection Results

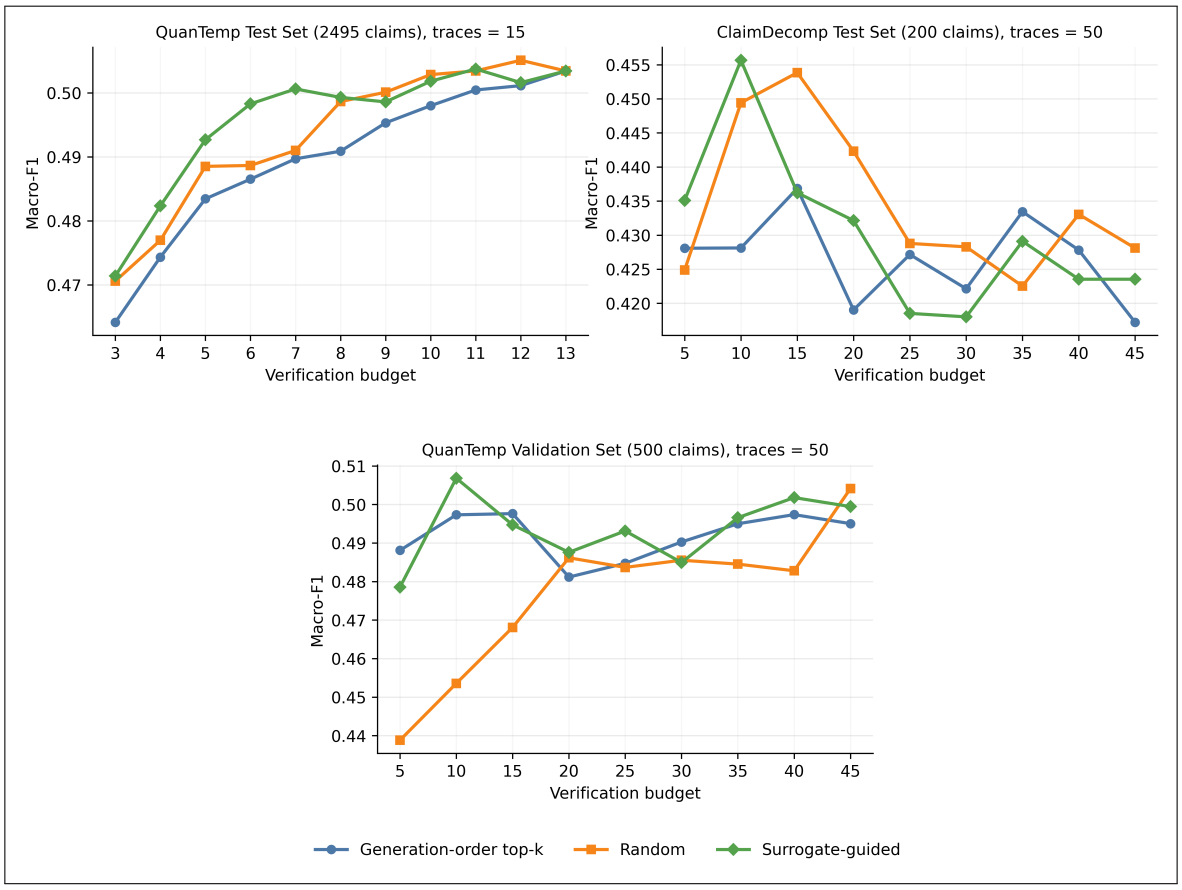



**Figure 5.2:** Macro-F1 performance across fixed verifier budgets $k$, comparing three selection strategies on QuanTemp test, QuanTemp validation, and ClaimDecomp test trace pools.

Figure 5.2 compares surrogate-guided selection against generation-order top-$k$ and random selection under fixed verifier budgets. Across all three trace pools, surrogate-guided selection is most beneficial at low verifier budgets, where only a small fraction of the available traces can be verified.

On the QuanTemp test set ($N = 15$), surrogate-guided selection consistently outperforms generation-order selection for budgets between $k = 3$ and $k = 11$. The largest gain is observed at $k = 6$, where Macro-F1 increases from approximately 0.487 to 0.499, corresponding to a gain of $+1.2$ points. Surrogate-guided selection reaches near-maximum performance using

fewer verifier calls, whereas generation-order selection requires substantially larger budgets to achieve similar results. As the budget approaches the full pool size, the performance of all methods starts to converge because most traces have already been verified.

The same trend is visible on the larger QuanTemp validation trace pool ($N = 50$). At $k = 10$, surrogate-guided selection reaches 0.507 Macro-F1, compared with 0.498 for generation-order selection and 0.454 for random selection. This corresponds to gains of approximately $+0.9$ and $+5.3$ Macro-F1 points, respectively. These results suggest that the surrogate is particularly useful when the candidate pool is large and the verifier budget is highly constrained.

On the ClaimDecomp test set ($N = 50$), surrogate-guided selection also performs strongly at low budgets, reaching its highest Macro-F1 of 0.456 at $k = 10$. This improves over generation-order selection by approximately $+2.8$ points and over random selection by $+0.6$ points. However, the trends are noisier than on QuanTemp, which is expected given the smaller size of the ClaimDecomp test set. Overall, the results show that surrogate-guided selection can prioritize useful traces early, improving verifier efficiency most clearly in the low-budget regime.

Appendix B.3 reports selected numeric values behind these curves and adds the verifier-score top-$k$ diagnostic reference. That reference uses verifier scores from the candidate pool, so it is useful as an upper-bound comparison for verifier-score information, not as a deployable budgeted selection method.

**Key Insights**

1. *Surrogate-guided selection is most useful when the verifier budget is small. On the QuanTemp test set, it outperforms generation-order selection between $k = 3$ and $k = 11$, with the largest gain at $k = 6$.*
2. *The surrogate can recover strong performance with fewer verifier calls. On the QuanTemp validation pool ($N = 50$), surrogate-guided selection reaches **0.507** Macro-F1 at $k = 10$, compared with **0.498** for generation-order selection and **0.454** for random selection.*
3. *Surrogate-guided selection also improves low-budget verification on ClaimDecomp. At $k = 10$, it reaches **0.456** Macro-F1, outperforming generation-order selection by approximately **2.8 percentage points** and random selection by **0.6 percentage points**.*

## 5.6. Ablation Studies

### 5.6.1. Verifier Model Choice

This ablation tests whether the efficiency gains observed in the main experiments also hold with a different verifier model. The generator is kept fixed as Llama 3.2-3B, while the verifier is changed from the LoRA fine-tuned Llama 3.2-3B verifier to DeepSeek-R1:7B. The experiment is conducted only on the QuanTemp test set because it is the larger dataset and is more representative of the main claim-verification task studied in this thesis.

**Table 5.8:** DeepSeek-R1:7B verifier ablation: method performance ($N = 15$).

| Method | T-F1 | F-F1 | C-F1 | M-F1 | W-F1 | Acc. |
|---|---|---|---|---|---|---|
| Exhaustive BoN (baseline) | **39.09** | **60.77** | 24.54 | 41.46 | 47.96 | 47.25 |
| Adaptive Stopping | 38.67 | 60.17 | 29.53 | **42.79** | 48.74 | **47.37** |
| Surrogate-Guided Selective Verification | 37.56 | 60.61 | **29.87** | 42.68 | **48.86** | 46.97 |

**Table 5.9:** DeepSeek-R1:7B verifier ablation: method efficiency ($N = 15$).

| Method | Avg traces /claim | Trace attempts | Missing-label traces | Verifier calls | Total operations | Operations reduction |
|---|---|---|---|---|---|---|
| Exhaustive BoN (baseline) | 15.00 | 37,425 | 313 | 37,112 | 74,537 | – |
| Adaptive Stopping | **5.14** | **12,819** | 125 | 12,694 | **25,513** | $-65.8\%$ |
| Surrogate-Guided Selective Verification | 6.01 | 15,005 | 127 | **12,255** | 27,260 | $-63.4\%$ |

**Table 5.10:** DeepSeek-R1:7B verifier ablation: latency, estimated total processing runtime, and estimated cost ($N = 15$). Runtime is computed from summed per-claim processing latency and corresponds to the estimated cost column.

| Method | Gen./claim (s) | Ver./claim (s) | Total/claim (s) | Runtime (h) | Cost (€) |
|---|---|---|---|---|---|
| Exhaustive BoN (baseline) | 17.60 | 4.97 | 22.58 | 15.65 | 0.978 |
| Adaptive Stopping | **5.69** | **1.48** | **7.18** | **4.97** | **0.311** |
| Surrogate-Guided Selective Verification | 6.80 | 1.76 | 8.60 | 5.96 | 0.372 |

Tables 5.8-5.10 show that changing the verifier substantially shifts the cost-performance trade-off. DeepSeek-R1:7B is cheaper to use in this setup, but it also gives lower verification performance than the LoRA fine-tuned verifier used in the main experiments. This difference is expected because the two verifier configurations are not equivalent: DeepSeek-R1:7B is used as a prompt-based verifier that returns a JSON score from an instruction prompt, whereas the main verifier is a LoRA fine-tuned consistency classifier.

With DeepSeek-R1:7B, Adaptive Stopping reaches 42.79 Macro-F1 and 47.37 accuracy at an estimated cost of €0.311. In comparison, Adaptive Stopping with the LoRA verifier reaches 49.41 Macro-F1 and 53.27 accuracy at €0.715. Thus, the DeepSeek-R1:7B setting reduces cost, but at a clear performance penalty.

Within the DeepSeek-R1:7B ablation itself, the relative method behaviour remains similar to the main results. Adaptive Stopping is the lowest-latency and lowest-cost configuration because it generates fewer traces, while Surrogate-Guided Selective Verification uses the fewest verifier calls. The surrogate run remains close to Adaptive Stopping in Macro-F1, with 42.68 compared with 42.79, and reduces verifier calls to 12,255 compared with 12,694 for Adaptive Stopping and 37,112 for Exhaustive BoN. These results suggest that verifier choice affects the absolute performance level, but the efficiency patterns of Adaptive Stopping and surrogate-guided verification remain visible.

### 5.6.2. Increased Trace Budget

This ablation evaluates whether increasing the maximum trace budget improves QuanTemp performance. For Exhaustive BoN, the $n$=15, $n$=20, and $n$=30 rows are computed by truncating the 40-trace run to the first $n$ traces and selecting the highest verifier-scored trace. Adaptive Stopping rows are computed from the saved 40-trace adaptive run using the same trace caps. Due to computational constraints, this ablation is limited to Exhaustive BoN and Adaptive Stopping. Running the surrogate-guided method across multiple larger trace budgets would require additional full inference runs, so the analysis focuses on the direct trace-budget comparison between full-budget decoding and adaptive stopping.

**Table 5.11:** QuanTemp performance as the maximum trace budget increases.

| Max traces | Method | T-F1 | F-F1 | C-F1 | M-F1 | W-F1 | Acc. |
|---|---|---|---|---|---|---|---|
| 15 | Exhaustive BoN (baseline) | 36.71 | **66.55** | **43.94** | **49.07** | **55.46** | **53.47** |
| 15 | Adaptive Stopping | **37.84** | 63.94 | 42.90 | 48.23 | 53.76 | 51.74 |
| 20 | Exhaustive BoN (baseline) | 37.76 | **66.83** | **45.67** | **50.09** | **56.24** | **54.39** |
| 20 | Adaptive Stopping | **38.34** | 64.01 | 43.48 | 48.61 | 54.04 | 52.11 |
| 30 | Exhaustive BoN (baseline) | 36.90 | **67.89** | **46.36** | **50.38** | **56.84** | **55.03** |
| 30 | Adaptive Stopping | **38.97** | 64.61 | 43.99 | 49.19 | 54.62 | 52.72 |
| 40 | Exhaustive BoN (baseline) | 37.73 | **68.06** | **47.03** | **50.94** | **57.26** | **55.47** |
| 40 | Adaptive Stopping | **39.24** | 64.44 | 44.37 | 49.35 | 54.67 | 52.76 |

*Note: The $N = 15$ rows in this ablation come from a saved 40-trace run truncated to the first 15 traces. They differ from the independent $N = 15$ main experiment because the LLM-generated traces are stochastic across runs.*

**Table 5.12:** QuanTemp computational efficiency as the maximum trace budget increases.

| Max traces | Method | Avg traces /claim | Trace attempts | Missing-label traces | Verifier calls | Total operations |
|---|---|---|---|---|---|---|
| 15 | Exhaustive BoN (baseline) | 15.00 | 37,425 | 301 | 37,124 | 74,549 |
| 15 | Adaptive Stopping | **9.28** | **23,156** | 170 | **22,986** | **46,142** |
| 20 | Exhaustive BoN (baseline) | 20.00 | 49,900 | 404 | 49,496 | 99,396 |
| 20 | Adaptive Stopping | **11.53** | **28,761** | 213 | **28,548** | **57,309** |
| 30 | Exhaustive BoN (baseline) | 30.00 | 74,850 | 610 | 74,240 | 149,090 |
| 30 | Adaptive Stopping | **15.94** | **39,777** | 285 | **39,492** | **79,269** |
| 40 | Exhaustive BoN (baseline) | 40.00 | 99,800 | 826 | 98,974 | 198,774 |
| 40 | Adaptive Stopping | **20.28** | **50,603** | 355 | **50,249** | **100,852** |

Table 5.11 shows that Exhaustive BoN performance largely plateaus as the trace budget increases: Macro-F1 increases only from 49.07 at $N = 15$ to 50.94 at $N = 40$, and accuracy rises from 53.47 to 55.47. However, Tables 5.12 and 5.13 show that this small improvement comes with a large increase in computation. Exhaustive BoN total operations grow from 74,549 to 198,774, a 166.6% increase, and estimated runtime grows from 20.32 to 54.21 hours. Adaptive Stopping remains consistently cheaper at every trace cap. At $N = 40$, it reaches 49.35 Macro-F1 with 100,852 operations and 25.90 hours of estimated runtime, recovering most of the Exhaustive BoN performance while using roughly half the operations and runtime. Overall, the ablation shows that increasing the trace budget does not necessarily

**Table 5.13:** QuanTemp latency, estimated total processing runtime, and estimated cost as the maximum trace budget increases. Runtime is computed from summed per-claim processing latency and corresponds to the estimated cost column.

| Max traces | Method | Gen./claim (s) | Ver./claim (s) | Total/claim (s) | Runtime (h) | Cost (€) |
|---|---|---|---|---|---|---|
| 15 | Exhaustive BoN (baseline) | 17.58 | 11.75 | 29.33 | 20.32 | 1.270 |
| 15 | Adaptive Stopping | **10.52** | **7.14** | **17.66** | **12.24** | **0.765** |
| 20 | Exhaustive BoN (baseline) | 23.45 | 15.66 | 39.11 | 27.10 | 1.694 |
| 20 | Adaptive Stopping | **12.82** | **8.86** | **21.68** | **15.02** | **0.939** |
| 30 | Exhaustive BoN (baseline) | 35.17 | 23.49 | 58.66 | 40.66 | 2.541 |
| 30 | Adaptive Stopping | **17.33** | **12.26** | **29.59** | **20.51** | **1.282** |
| 40 | Exhaustive BoN (baseline) | 46.89 | 31.31 | 78.21 | 54.21 | 3.388 |
| 40 | Adaptive Stopping | **21.77** | **15.60** | **37.37** | **25.90** | **1.619** |

produce proportional performance gains. This strengthens the case for Adaptive Stopping, since the method avoids spending the full budget when additional traces are unlikely to justify their computational cost.

### 5.6.3. Unbounded Adaptive Stopping

An unbounded Adaptive Stopping setting was also tested, in which the system continued generating and verifying traces until the adaptive margin criterion was satisfied rather than stopping at a fixed maximum trace budget. This setting was not helpful in several cases. The main failure mode is that some claims never produce a sufficiently separated top verifier score. For example, one QuanTemp claim reached the capped 15-trace limit with a best–second verifier margin of only 0.0099, far below the stopping threshold of 0.15. When the same claim was run without the fixed cap, it continued to 1,827 verified traces and still had a margin of only 0.0003, so the stopping criterion was never triggered.

This behaviour reflects a limitation of margin-based Adaptive Stopping rather than an implementation error. If the generator repeatedly produces many plausible traces with similar labels or similarly high verifier scores, the verifier-score distribution remains flat near the top. In that situation, additional generation does not create a confident separation; it instead keeps producing near-ties that prevent the margin rule from firing. The unbounded setting can therefore spend extremely large amounts of computation on claims whose generation quality and verifier calibration do not support a decisive margin. Adaptive Stopping is most effective when used with a maximum trace limit: the margin rule can stop easy cases early, while the cap prevents ambiguous or low-separation cases from running indefinitely.

**Key Insights**

1. *Verifier choice changes the absolute cost-performance level. With DeepSeek-R1:7B, Adaptive Stopping is cheaper, with an estimated cost of* ***€0.311****, but its performance drops to* ***42.79*** *Macro-F1 compared with* ***49.41*** *Macro-F1 for the LoRA verifier. This shows that the verifier formulation strongly affects final verification quality.*
2. *Increasing the trace budget gives diminishing performance returns. Exhaustive BoN Macro-F1 rises only from* ***49.07*** *at $N = 15$ to* ***50.94*** *at $N = 40$, while total operations increase by* ***166.6%*** *and estimated runtime increases by* ***166.8%****. This suggests that simply generating more traces is not an efficient way to improve performance.*
3. *Adaptive Stopping remains important when larger budgets are available, but it needs a maximum cap. At $N = 40$, Adaptive Stopping recovers most of the Exhaustive BoN performance with* ***49.35*** *Macro-F1 while using roughly half the operations and runtime. The unbounded setting shows why the cap is necessary: ambiguous claims can keep generating traces without ever satisfying the margin threshold.*

## 5.7. Summary of Findings by Research Question

Table 5.14 summarizes how each research question is addressed by the empirical results in this chapter, linking the evaluation setting to the main finding and supporting evidence.

**Table 5.14:** Mapping between research questions, evaluation settings, and main findings.

| Evaluation | Evaluation setting | Main finding | Key evidence |
|---|---|---|---|
| Baseline selection | Top-1 decoding, self-consistency, and Exhaustive BoN on QuanTemp and ClaimDecomp. | Exhaustive BoN is the strongest overall high-compute baseline. | Exhaustive BoN reaches 51.49 Macro-F1 on QuanTemp and 41.27 Macro-F1 on ClaimDecomp. |
| RQ1 | Exhaustive BoN vs. Adaptive Stopping with $N = 15$ on QuanTemp and ClaimDecomp. | Adaptive Stopping substantially reduces inference cost while maintaining comparable verification performance. | Operations decrease by 38.8% on QuanTemp and 51.8% on ClaimDecomp; Macro-F1 changes by $-2.08$pp and $+0.58$pp, respectively. |
| RQ2 | Exhaustive BoN, Adaptive Stopping, and Surrogate-Guided Selective Verification with $N = 15$. | Surrogate guidance further reduces verifier usage while remaining in a comparable performance range. | Verifier calls decrease by 44.5% relative to Exhaustive BoN on QuanTemp and 53.7% on ClaimDecomp; additional savings relative to Adaptive Stopping are 9.4% and 4.0%. |
| RQ3 | Fixed verifier budgets on QuanTemp test ($N = 15$), ClaimDecomp test ($N = 50$), and QuanTemp validation ($N = 50$). | Surrogate-guided selection is most useful in low-budget verifier-allocation settings. | At $k = 10$, surrogate-guided selection reaches 50.68 Macro-F1 on QuanTemp validation and 45.57 Macro-F1 on ClaimDecomp. |
| Ablations | Verifier-model, increased trace-budget, and unbounded Adaptive Stopping analyses on QuanTemp. | Verifier choice affects absolute performance, larger exhaustive budgets have diminishing returns, and Adaptive Stopping needs a maximum cap. | DeepSeek-R1:7B lowers cost but reduces Macro-F1; Exhaustive BoN operations increase by 166.6% from $N = 15$ to $N = 40$. |

# 6

# Conclusion

## 6.1. Conclusion

This thesis studies the optimization of verifier-guided test-time scaling for fact checking through two strategies, Verifier-Guided Adaptive Stopping and Surrogate-Guided Selective Verification. These strategies address a central limitation of Exhaustive Best-of-N (BoN), which improves verification by generating multiple reasoning traces and selecting the trace most strongly supported by a verifier, but applies the same fixed generation and verification budget to every claim. The baseline results show why this trade-off matters. On QuanTemp, Exhaustive BoN improves Macro-F1 from 39.93 to 51.49 and accuracy from 41.88% to 55.71% compared with top-1 decoding, confirming that multi-trace verifier-guided reasoning is useful for numerical and evidence-sensitive claims. However, this performance comes from generating 15 traces per claim and verifying every valid trace, which makes the method expensive and motivates adaptive allocation of test-time computation.

The first strategy, Adaptive Stopping, uses verifier-score margins as a signal of prediction stability. Instead of always generating the full trace budget, it stops once the verifier shows a sufficiently strong preference for one label. Empirically, this gives the clearest end-to-end efficiency gain. On QuanTemp, Adaptive Stopping reduces total generation-plus-verification operations by 38.8%, generation tokens by 38.9%, and estimated GPU energy cost from €1.232 to €0.715, while maintaining performance close to the Exhaustive BoN baseline, with a 2.08 percentage-point run-level difference in Macro-F1. On ClaimDecomp, it reduces operations by 51.8% and generation tokens by 51.9%, while remaining comparable to the 200-claim Exhaustive BoN evaluation under the 15-trace cap. Since the methods generate independent stochastic trace sets, these small performance differences should not be interpreted as deterministic improvements or degradations. The more robust result is that Adaptive Stopping preserves much of the baseline performance while using substantially less inference cost and latency.

The second strategy, Surrogate-Guided Selective Verification, targets the verifier side of the pipeline. It uses a lightweight online surrogate to estimate which generated traces are most worth sending to the verifier. The empirical results show that this selective allocation reduces verifier usage while maintaining comparable run-level performance. Relative to Exhaustive

BoN, verifier calls decrease by 44.5% on QuanTemp and 53.7% on ClaimDecomp. Compared with Adaptive Stopping, the method achieves further reductions of 9.4% and 4.0%, respectively. The main drawback is that verifier-call savings do not automatically translate into lower total latency or cost. Because the surrogate may require additional trace generation and selection overhead, Adaptive Stopping remains the lowest-cost and lowest-latency method in the main LoRA-verifier setting. Thus, the surrogate is most useful when verifier calls are the main bottleneck, rather than as a universal replacement for Adaptive Stopping.

To provide a more complete evaluation, this thesis also analyzes the surrogate as a selection mechanism under fixed verifier budgets. The results show that trace selection matters when only a subset of generated traces can be verified. Surrogate guidance improves Macro-F1 by up to 1.2 percentage points on the QuanTemp test set and also outperforms both generation-order and random selection at key low-budget settings on QuanTemp validation and ClaimDecomp. These gains indicate that online surrogate learning is worth the added complexity and overhead because it helps prioritize more useful traces and provides a promising basis for developing stronger selection algorithms.

Overall, the results support the view that verifier-guided fact checking should be evaluated as a cost-performance problem, not only as a prediction problem. Exhaustive BoN remains a strong high-compute baseline, but it allocates computation uniformly across claims. Adaptive Stopping provides the most reliable end-to-end cost reduction because it avoids unnecessary generation and verification while maintaining performance close to the exhaustive baseline. Surrogate-Guided Selective Verification is most useful as a verifier-allocation mechanism when verifier calls are scarce or expensive. The main conclusion is that test-time computation should be allocated dynamically, using intermediate verifier feedback and trace-level signals to decide when additional reasoning is useful and when further computation is unlikely to change the final verdict.

## 6.2. Limitations and Assumptions

### Dependence on a Maximum Trace Bound

Adaptive Stopping is defined relative to a fixed upper bound on generation, such as $N = 15$. It can decide to stop earlier than this cap, but it does not answer whether more than $N$ traces would improve the prediction. Therefore, the measured savings and performance trade-off depend on the chosen maximum trace budget. At the same time, the unbounded experiment in Section 5.6.3 shows why this cap is practically necessary: some ambiguous claims never produce a sufficiently large verifier-score margin, even after many additional traces, so an uncapped setting can spend excessive computation waiting for a margin that may not emerge.

### Stochastic Run-Level Comparisons

Each method generates its own set of stochastic LLM traces, so small performance differences should be interpreted as run-level outcomes rather than deterministic method improvements. This is especially important for the smaller 200-claim ClaimDecomp setting, where normal

generation variance can have a larger effect on Macro-F1.

### Limited Dataset Scope

The evaluation focuses on QuanTemp, with additional experiments on a smaller completed ClaimDecomp setting. These datasets cover numerical, temporal, and decompositional claim verification, but they do not cover all real-world fact-checking settings. The results may differ for multilingual claims, longer evidence contexts, or other domains such as health, law, and finance.

### Verifier Score Reliability

The methods assume that verifier scores are useful proxies for reasoning-trace quality. If the verifier assigns high scores to fluent but unsupported justifications, misses numerical inconsistencies, or is biased toward particular labels, Adaptive Stopping and surrogate-guided selection may make confident but incorrect decisions. This thesis evaluates final verdict performance, but it does not separately evaluate the faithfulness of reasoning or verifier calibration.

### Model and Hyperparameter Dependence

The results depend on the generator, verifier, prompts, verified-margin threshold, warm-up size, surrogate selection parameters, and trace budget. The DeepSeek-R1:7B ablation shows that changing the verifier can alter both predictive quality and the cost-performance frontier. Thresholds and allocation policies may therefore require recalibration for different model, prompt, or deployment settings.

## 6.3. Future Work

### Richer Trace Features

The current surrogate uses lightweight trace-level features so that selection remains inexpensive. Future work could explore richer features that capture deeper semantic, numerical, and evidence-grounding signals, such as claim-evidence alignment, justification faithfulness, trace diversity, and contradiction patterns across traces. These features may help the surrogate better estimate which traces are likely to receive high verifier scores, especially for ambiguous or numerically complex claims.

### Non-linear Surrogate Models

This thesis uses a simple online linear surrogate, which keeps the method interpretable and cheap but limits the kinds of relationships it can learn. Future work could test non-linear surrogate models, such as small neural networks, tree-based models, or kernel methods, to capture interactions between trace features. The main challenge would be preserving the low overhead that makes Surrogate-Guided Selective Verification useful in the first place.

### Confidence Calibration and Conformal Prediction

The Adaptive Stopping rule relies on verifier-score margins as a proxy for prediction stability, but verifier scores are not guaranteed to be calibrated. Future work could calibrate verifier con-

fidence using validation data and study whether calibrated scores lead to more reliable stopping decisions. Conformal prediction could also be explored as a framework for uncertainty-aware decision rules, for example by allowing the system to abstain, continue generation, or request more verification when confidence is insufficient.

#### Hyperparameter Sensitivity

The results depend on design choices such as the verified-margin threshold, trace budgets, warm-up size, and surrogate selection parameters. Future work should systematically analyze these hyperparameters instead of treating them as fixed configuration choices. A more principled approach could tune them for a target cost-performance constraint or learn them from validation data, making the adaptive pipeline less dependent on manually selected values.

## 6.4. AI Disclosure

In line with TU Delft's publishing policies,[1] I acknowledge the use of AI tools, including Claude Code, ChatGPT and Codex to support the writing and preparation of this thesis. These tools were used for language refinement, drafting assistance, LaTeX editing support, and code-related assistance during experimentation and analysis. AI tools were used as assistive tools only; the research design, implementation choices, experiments, interpretation of results, and final submitted text remain my own responsibility.

All AI-assisted text and code suggestions were reviewed, edited, and checked against the implemented experiments and recorded results before inclusion. Some visual material and figure drafts were prepared or refined with the help of AI-based tools, and icons used in figures were created by contributors from `www.flaticon.com`. Any AI-generated or AI-assisted material was manually reviewed for accuracy, relevance, and consistency with the thesis.

[1] TU Delft publishing policies: `https://www.tudelft.nl/library/actuele-themas/open-publishing/about/policies`

# References


[1] Soroush Vosoughi, Deb Roy, and Sinan Aral. “The spread of true and false news online”. In: *science* 359.6380 (2018), pp. 1146–1151.

[2] Kai Shu et al. “Fake news detection on social media: A data mining perspective”. In: *ACM SIGKDD explorations newsletter* 19.1 (2017), pp. 22–36.

[3] Andreas Vlachos and Sebastian Riedel. “Fact checking: Task definition and dataset construction”. In: *Proceedings of the ACL 2014 workshop on language technologies and computational social science*. 2014, pp. 18–22.

[4] Xia Zeng, Amani S Abumansour, and Arkaitz Zubiaga. “Automated fact-checking: A survey”. In: *Language and Linguistics Compass* 15.10 (2021), e12438.

[5] Zhijiang Guo, Michael Schlichtkrull, and Andreas Vlachos. “A survey on automated fact-checking”. In: *Transactions of the association for computational linguistics* 10 (2022), pp. 178–206.

[6] Pepa Atanasova et al. “Generating Fact Checking Explanations”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020, pp. 7352–7364. doi: 10.18653/v1/2020.acl-main.656. url: https://aclanthology.org/2020.acl-main.656/.

[7] Neema Kotonya and Francesca Toni. “Explainable Automated Fact-Checking for Public Health Claims”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, 2020, pp. 7740–7754. doi: 10.18653/v1/2020.emnlp-main.623. url: https://aclanthology.org/2020.emnlp-main.623/.

[8] Jifan Chen et al. “Generating literal and implied subquestions to fact-check complex claims”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 2022, pp. 3495–3516.

[9] Ivan Vykopal et al. *Generative Large Language Models in Automated Fact-Checking: A Survey*. 2024. doi: 10.48550/arXiv.2407.02351. arXiv: 2407.02351 [cs.CL]. url: https://arxiv.org/abs/2407.02351.

[10] Dorian Quelle and Alexandre Bovet. “The Perils and Promises of Fact-Checking with Large Language Models”. In: *Frontiers in Artificial Intelligence* 7 (2024), p. 1341697. doi: 10.3389/frai.2024.1341697. url: https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2024.1341697/full.

[11] Liangming Pan et al. “Fact-Checking Complex Claims with Program-Guided Reasoning”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 6981–7004. doi: 10.18653/v1/2023.acl-long.386. url: https://aclanthology.org/2023.acl-long.386/.

[12] Yichen Jiang et al. “HoVer: A dataset for many-hop fact extraction and claim verification”. In: *arXiv preprint arXiv:2011.03088* (2020).

[13] Michael Schlichtkrull, Zhijiang Guo, and Andreas Vlachos. “Averitec: A dataset for real-world claim verification with evidence from the web”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 65128–65167.

[14] Abhijit Anand, Avishek Anand, Vinay Setty, et al. “Quantemp: A real-world open-domain benchmark for fact-checking numerical claims”. In: *arXiv preprint arXiv:2403.17169* (2024).

[15] Mubashara Akhtar et al. “Exploring the numerical reasoning capabilities of language models: A comprehensive analysis on tabular data”. In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. 2023, pp. 15391–15405.

[16] Jared Kaplan et al. “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361* (2020).

[17] Jordan Hoffmann et al. “Training compute-optimal large language models”. In: *arXiv preprint arXiv:2203.15556* 10 (2022).

[18] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837.

[19] Xuezhi Wang et al. “Self-consistency improves chain of thought reasoning in language models”. In: *arXiv preprint arXiv:2203.11171* (2022).

[20] Charlie Snell et al. “Scaling llm test-time compute optimally can be more effective than scaling model parameters”. In: *arXiv preprint arXiv:2408.03314* (2024).

[21] Karl Cobbe et al. “Training verifiers to solve math word problems”. In: *arXiv preprint arXiv:2110.14168* (2021).

[22] Hunter Lightman et al. “Let’s verify step by step”. In: *The twelfth international conference on learning representations*. 2023.

[23] Liyan Tang, Philippe Laban, and Greg Durrett. “Minicheck: Efficient fact-checking of llms on grounding documents”. In: *arXiv preprint arXiv:2404.10774* (2024).

[24] Zhengbao Jiang et al. “Active retrieval augmented generation”. In: *Proceedings of the 2023 conference on empirical methods in natural language processing*. 2023, pp. 7969–7992.

[25] Soyeong Jeong et al. “Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity”. In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 2024, pp. 7036–7050.

[26] James Thorne et al. “FEVER: a large-scale dataset for fact extraction and VERification”. In: *arXiv preprint arXiv:1803.05355* (2018).

[27] Samuel Bowman et al. “A large annotated corpus for learning natural language inference”. In: *Proceedings of the 2015 conference on empirical methods in natural language processing*. 2015, pp. 632–642.

[28] Rami Aly et al. “Feverous: Fact extraction and verification over unstructured and structured information”. In: *arXiv preprint arXiv:2106.05707* (2021).

[29] Primakov Chungkham et al. “Think Right, Not More: Test-Time Scaling for Numerical Claim Verification”. In: *Empirical Methods in Natural Language Processing (EMNLP) 2025*. Association for Computational Linguistics. 2025, pp. 24345–24363.

[30] Mohammed Abdul Khaliq et al. “Ragar, your falsehood radar: Rag-augmented reasoning for political fact-checking using multimodal large language models”. In: *Proceedings of the Seventh Fact Extraction and VERification Workshop (FEVER)*. 2024, pp. 280–296.

[31] Freda Shi et al. “Language models are multilingual chain-of-thought reasoners”. In: *arXiv preprint arXiv:2210.03057* (2022).

[32] Miles Turpin et al. “Language models don’t always say what they think: Unfaithful explanations in chain-of-thought prompting”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 74952–74965.

[33] Jia Li et al. “Structured chain-of-thought prompting for code generation”. In: *ACM Transactions on Software Engineering and Methodology* 34.2 (2025), pp. 1–23.

[34] Zheng Zhao et al. “Verifying Chain-of-Thought Reasoning via Its Computational Graph”. In: *arXiv preprint arXiv:2510.09312* (2025).

[35] Jingwei Ni et al. “Reasoning with Confidence: Efficient Verification of LLM Reasoning Steps via Uncertainty Heads”. In: *arXiv preprint arXiv:2511.06209* (2025).

[36] Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. “On the self-verification limitations of large language models on reasoning and planning tasks”. In: *The Thirteenth International Conference on Learning Representations*. 2025. url: https://openreview.net/forum?id=4O0v4s3IzY.

[37] Bartosz Piotrowski et al. “Lightweight Latent Verifiers for Efficient Meta-Generation Strategies”. In: *arXiv preprint arXiv:2504.16760* (2025).

[38] Shunyu Yao et al. “Tree of thoughts: Deliberate problem solving with large language models”. In: *Advances in neural information processing systems* 36 (2023), pp. 11809–11822.

[39] Kun Li et al. “Decoding on graphs: Faithful and sound reasoning on knowledge graphs through generation of well-formed chains”. In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2025, pp. 24349–24364.

[40] Jonathan Uesato et al. “Solving math word problems with process-and outcome-based feedback”. In: *arXiv preprint arXiv:2211.14275* (2022).

[41] Chengsong Huang et al. “Efficient test-time scaling via self-calibration”. In: *arXiv preprint arXiv:2503.00031* (2025).

[42] Jinyan Su et al. “Between underthinking and overthinking: An empirical study of reasoning length and correctness in llms”. In: *arXiv preprint arXiv:2505.00127* (2025).

[43] Abhinav Kumar et al. “Overthink: Slowdown attacks on reasoning llms”. In: *arXiv preprint arXiv:2502.02542* (2025).

[44] Michael Hassid et al. “Don’t Overthink it. Preferring Shorter Thinking Chains for Improved LLM Reasoning”. In: *arXiv preprint arXiv:2505.17813* (2025).

[45] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. “Branchynet: Fast inference via early exiting from deep neural networks”. In: *2016 23rd international conference on pattern recognition (ICPR)*. IEEE. 2016, pp. 2464–2469.

[46] Hanzhang Hu et al. “Learning anytime predictions in neural networks via adaptive loss balancing”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 3812–3821.

[47] Hanzhang Hu et al. “Anytime neural network: a versatile trade-off between computation and accuracy”. In: (2018).

[48] Metod Jazbec et al. “Towards anytime classification in early-exit architectures by enforcing conditional monotonicity”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 56138–56168.

[49] Samuel Lewis-Lim et al. “Can Confidence Estimates Decide When Chain-of-Thought Is Necessary for LLMs?” In: *arXiv preprint arXiv:2510.21007* (2025).

[50] Tingxu Han et al. “Token-budget-aware llm reasoning”. In: *Findings of the Association for Computational Linguistics: ACL 2025*. 2025, pp. 24842–24855.

[51] Kyle Montgomery et al. “Budget-aware Test-time Scaling via Discriminative Verification”. In: *arXiv preprint arXiv:2510.14913* (2025).

[52] Zhenwen Liang et al. “Improving llm reasoning through scaling inference computation with collaborative verification”. In: *arXiv preprint arXiv:2410.05318* (2024).

[53] Volodymyr Kuleshov and Doina Precup. “Algorithms for multi-armed bandit problems”. In: *arXiv preprint arXiv:1402.6028* (2014).

[54] Sampath Kannan et al. “A smoothed analysis of the greedy algorithm for the linear contextual bandit problem”. In: *Advances in neural information processing systems* 31 (2018).

[55] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. “Finite-time analysis of the multi-armed bandit problem”. In: *Machine learning* 47.2 (2002), pp. 235–256.

[56] Gabriele Farina, Robin Schmucker, and Tuomas Sandholm. “Bandit linear optimization for sequential decision making and extensive-form games”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 6. 2021, pp. 5372–5380.

[57] Marta Soare, Alessandro Lazaric, and Rémi Munos. “Best-arm identification in linear bandits”. In: *Advances in neural information processing systems* 27 (2014).

[58] Niranjan Srinivas et al. “Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design”. In: *Proceedings of the 27th International Conference on Machine Learning*. 2010, pp. 1015–1022.

[59] Dongruo Zhou, Lihong Li, and Quanquan Gu. “Neural Contextual Bandits with UCB-based Exploration”. In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR. 2020, pp. 11492–11502.

[60] Melda Korkut and Andrew Li. “Disposable linear bandits for online recommendations”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 5. 2021, pp. 4172–4180.

[61] Kiran Purohit et al. *CASE: Challenger Arm Sampling for Efficient In-Context Reasoning*. 2025. url: https://openreview.net/forum?id=qvwcK4Uz8z.

[62] Mandeep Rathee et al. “Breaking the lens of the telescope: Online relevance estimation over large retrieval sets”. In: *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2025, pp. 2287–2297.

[63] Jacob Benesty and Tomas Gänsler. “New Insights into the RLS Algorithm”. In: *EURASIP Journal on Applied Signal Processing* 2004.3 (2004), pp. 331–339. doi: 10.1155/S1110865704310188. url: https://asp-eurasipjournals.springeropen.com/articles/10.1155/S1110865704310188.

[64] Aaron Grattafiori et al. “The llama 3 herd of models”. In: *arXiv preprint arXiv:2407.21783* (2024).

[65] DeepSeek-AI et al. “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning”. In: *arXiv preprint arXiv:2501.12948* (2025). url: https://arxiv.org/abs/2501.12948.

[66] Nils Reimers and Iryna 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)*. 2019, pp. 3982–3992.

# A

# Experimental Setup

This appendix consolidates the experimental details used across the main results chapter. Chapter 4 describes the design choices in prose; this appendix provides a compact reference for datasets, model components, experimental variants, hyperparameters, and the generator prompt template.

## A.1. Datasets and Evaluation Splits

Table A.1 summarizes the dataset splits used in the experiments. QuanTemp is the primary benchmark and ClaimDecomp is used as a secondary dataset for checking whether the adaptive and surrogate-guided methods remain useful beyond numerical claim verification.

**Table A.1:** Dataset splits and evidence sources used in the experiments.

| Dataset | Split / setting | Claims | Evidence source | Use in this thesis |
|---|---|---|---|---|
| QuanTemp | Test | 2,495 | BM25 candidates re-ranked with sentence-transformer similarity; top three passages used as evidence. | Main evaluation for baseline decoding, Adaptive Stopping, surrogate-guided verification, fixed-budget selection, and ablations. |
| QuanTemp | Validation subset | 500 | Same retrieval and re-ranking pipeline as the test split. | Stratified higher-trace setting for the fixed-budget selection analysis. |
| ClaimDecomp | Test | 200 | Decompositional evidence fields provided with the dataset. | Secondary evaluation for Adaptive Stopping, surrogate-guided verification, and fixed-budget selection under complex claim decomposition. |

For QuanTemp, the original labels are mapped to three classes: Supports, Refutes, and Conflicting. The full QuanTemp split sizes are 9,935 training claims, 3,084 validation claims, and 2,495 test claims. The test split contains 474 true claims, 1,423 false claims, and 598 conflicting claims. The ClaimDecomp test split contains 60 true claims, 91 false claims, and 49 conflicting claims.

## A.2. Model and Pipeline Components

Table A.2 lists the main model components. Unless otherwise stated, all completed main experiments use Llama 3.2 3B as the generator and a LoRA fine-tuned Llama 3.2 3B verifier.

**Table A.2:** Model and pipeline components used for generation, verification, and semantic similarity.

| Component | Model / implementation | Role and configuration |
|---|---|---|
| Generator | Llama 3.2 3B via Ollama | Produces candidate reasoning traces. Each trace uses a maximum output length of 512 tokens, top-$p = 1.0$, and a balanced temperature schedule from 0.30 to 0.70 in steps of 0.05. |
| Main verifier | LoRA fine-tuned Llama 3.2 3B | Scores each claim, predicted verdict, and generated justification as a binary consistency classifier. LoRA rank is $r = 8$ with scaling parameter $\alpha = 16$. Scores are converted with a sigmoid function. |
| Verifier ablation | DeepSeek-R1:7B via Ollama | Prompt-based verifier used only in the verifier-model ablation. It returns a structured score in $[0, 1]$, rather than using a task-specific classifier head. |
| Embedding model | `paraphrase-MiniLM-L6-v2` | Used for semantic re-ranking of QuanTemp evidence candidates and for selecting semantically similar few-shot examples during prompt construction. |

## A.3. Experimental Variants

Table A.3 summarizes the experimental variants. The main comparisons use a maximum trace budget of $N = 15$. Additional fixed-budget and increased-trace analyses use larger candidate pools where saved runs are available.

**Table A.3:** Experimental variants used across the research questions and ablations.

| Experiment | Datasets | Trace setting | Comparison |
|---|---|---|---|
| Baseline performance comparison | QuanTemp, ClaimDecomp | $N = 15$ | Top-1 decoding, self-consistency, and Exhaustive BoN. |
| Verifier-Guided Adaptive Stopping | QuanTemp, ClaimDecomp | $N = 15$ | Exhaustive BoN vs. Verifier-Guided Adaptive Stopping. |
| Surrogate-Guided Selective Verification | QuanTemp, ClaimDecomp | $N = 15$ | Exhaustive BoN, Adaptive Stopping, and Surrogate-Guided Selective Verification. |
| Fixed-budget selection | QuanTemp test, ClaimDecomp test, QuanTemp validation subset | 15-trace and 50-trace candidate pools | Generation-order top-$k$, random selection with seed 20, and surrogate-guided selection under fixed verifier budgets. |
| Verifier ablation | QuanTemp test | $N = 15$ | Main LoRA verifier vs. DeepSeek-R1:7B prompt-based verifier. |
| Trace-budget ablation | QuanTemp test | $N \in \{15, 20, 30, 40\}$ | Exhaustive BoN and Adaptive Stopping under larger trace caps. |

## A.4. Generator Prompt Template

The generator prompt is constructed from semantically similar few-shot examples, subquestions where available, and retrieved evidence. The model is instructed to produce one justification and one final label for the full claim.

**Generator Prompt Template**

```
Here are examples of fact-checking:
[Claim]: <example claim 1>
[Subquestions + Evidence]
- Q: <sub-question 1> Evidence: <evidence doc 1>
```

```
- Q: <sub-question 2> Evidence: <evidence doc 2>
[Label]: <SUPPORTS|REFUTES|CONFLICTING>
... (further few-shot examples) ...

Following given examples, for the given claim, given
sub-questions and evidence use information from them to
fact check the claim and additionally paying attention
to numerical spans in claim and evidence and output the
label by performing entailment. Use the sub-questions to
guide your reasoning. Do not give [Label] for each
sub-question. Classify the entire claim as strictly one
of: SUPPORTS, REFUTES or CONFLICTING.

[Claim]: "<current claim>"
[Evidence]: "<current evidence>"

Generate [Justification]: and [Label]: in the end for
the whole claim. [Label]: must be strictly one of
SUPPORTS, REFUTES or CONFLICTING.
[Justification]:
[Label]:
```

## A.5. DeepSeek-R1:7B Verifier Prompt

DeepSeek-R1:7B Prompt-Based Verifier

```
SYSTEM_PROMPT = """You are a strict fact verification judge.

Your job is to evaluate whether the justification correctly supports the GIVEN verdict for the
    claim.

Verdict meanings:
- SUPPORTS: the justification must clearly show that the claim is true based on the evidence.
- REFUTES: the justification must clearly show that the claim is false or contradicted by the
    evidence.
- CONFLICTING: the justification must clearly show that the evidence is mixed, insufficient,
    ambiguous, or does not allow a definite support/refute decision.

Scoring rules:
- 1.0 = the justification strongly and correctly supports the given verdict
- 0.75 = mostly supports the given verdict, but has minor weakness
- 0.5 = partially supports the given verdict, but important problems remain
- 0.25 = weak match to the given verdict
- 0.0 = does not support the given verdict

Important:
- A justification that supports a different verdict must get 0.0 or 0.25.
- A vague or hedged justification should not get a high score.
- If the verdict is CONFLICTING, only give a high score if the justification explicitly explains
    why the evidence is mixed, ambiguous, or insufficient.
```

```
21 - If the verdict is SUPPORTS or REFUTES, do not reward a justification that says "partially
       supported", "not entirely accurate", "mixed", or "unclear".
22
23 Return only JSON:
24 {"score": 0.0}"""
```

# B

# Additional Quantitative Results

This appendix provides additional quantitative results supporting Chapter 5. Sections B.1 and B.2 extend the analysis of surrogate-guided verification by reporting a claim-matched trace-bucket comparison and a detailed runtime breakdown. Section B.3 supports the fixed-budget verifier-allocation analysis by reporting selected Macro-F1 values and the verifier-score top-(k) diagnostic reference.

## B.1. Trace-Bucket Comparison for Surrogate Guidance

Table B.1 groups QuanTemp and ClaimDecomp claims by the number of traces used by Adaptive Stopping or Surrogate-Guided Selective Verification. For each bucket, the table reports both the adaptive or surrogate-guided method Macro-F1 and the Exhaustive BoN Macro-F1 on the same claim subset. This makes the comparison claim-matched rather than comparing each method on a different bucket definition.

**Table B.1:** Macro-F1 by adaptive/surrogate trace-usage bucket. Exhaustive BoN is evaluated on the same claim subset for each row.

| Bucket method | Traces used | Claims | Method Macro-F1 | Exh. Macro-F1 |
|---|---|---|---|---|
| **QuanTemp** | | | | |
| Adaptive | 1–5 | 1299 | 45.16 | **49.58** |
| Adaptive | 6–10 | 67 | **39.99** | 37.43 |
| Adaptive | 11–15 | 1129 | 47.55 | **49.33** |
| Surrogate | 1–5 | 575 | **42.92** | 42.36 |
| Surrogate | 6–10 | 796 | 47.09 | **51.98** |
| Surrogate | 11–15 | 1124 | 45.96 | **48.08** |
| **ClaimDecomp** | | | | |
| Adaptive | 1–5 | 136 | 35.27 | **35.42** |
| Adaptive | 6–10 | 6 | **86.67** | 48.89 |
| Adaptive | 11–15 | 58 | 40.74 | **41.95** |
| Surrogate | 1–5 | 69 | 31.79 | **33.11** |
| Surrogate | 6–10 | 69 | **48.96** | 44.10 |
| Surrogate | 11–15 | 62 | **44.72** | 35.48 |

The ClaimDecomp bucket results should be interpreted descriptively because the dataset contains only 200 claims. In particular, the Adaptive Stopping 6–10 trace bucket contains six claims, making its Macro-F1 sensitive to individual predictions.

## B.2. Runtime and Cost Effects of Surrogate Guidance

Table B.2 decomposes the wall-clock runtime for the surrogate-guidance comparison into recorded generation time, recorded verifier time, and residual "other" time. The residual is computed as total wall-clock runtime minus generation runtime minus verifier runtime. It includes shared pipeline overhead such as evidence retrieval, prompt assembly, checkpointing, logging, and file I/O; for the surrogate-guided run, it also includes surrogate-side feature construction and RLS scoring. Costs use the same GPU rate used throughout the results chapter, €0.0625 per hour.

**Table B.2:** Wall-clock runtime and cost breakdown for surrogate-guided verification. Other runtime is the residual after subtracting recorded generation and verifier time from total wall-clock runtime.

| Dataset | Method | Gen. h | Ver. h | Other h | Total h | Gen. cost | Ver. cost | Other cost | Run cost |
|---|---|---|---|---|---|---|---|---|---|
| QuanTemp | Exhaustive BoN | 11.34 | 8.37 | 0.35 | 20.06 | 0.709 | 0.523 | 0.022 | 1.254 |
| QuanTemp | Adaptive Stopping Surrogate-Guided | 6.44 | 4.99 | 0.25 | 11.68 | 0.403 | 0.312 | 0.016 | 0.730 |
| QuanTemp | Selective Verification | 7.85 | **4.52** | 0.58 | 12.95 | 0.491 | **0.282** | 0.036 | 0.810 |
| ClaimDecomp | Exhaustive BoN | 0.76 | 0.65 | 0.03 | 1.44 | 0.047 | 0.041 | 0.002 | 0.090 |
| ClaimDecomp | Adaptive Stopping Surrogate-Guided | 0.42 | 0.38 | 0.05 | 0.84 | 0.026 | 0.024 | 0.003 | 0.053 |
| ClaimDecomp | Selective Verification | 0.51 | **0.30** | 0.05 | 0.86 | 0.032 | **0.019** | 0.003 | 0.054 |

Table B.3 isolates the surrogate-guided rows from Table B.2. The surrogate-side processing was not timed as a separate component in the saved metrics. Therefore, the residual "other" runtime gives a conservative upper bound on surrogate feature construction and RLS scoring plus shared non-generation, non-verifier pipeline overhead. The actual cost of the RLS prediction/update alone is smaller than this bound.

**Table B.3:** Conservative upper bound on linear surrogate overhead. The residual includes shared pipeline overhead plus surrogate-side feature construction and RLS scoring.

| Dataset | Upper-bound time (h) | Upper-bound cost (€) | Interpretation |
|---|---|---|---|
| QuanTemp | 0.58 | 0.036 | Actual RLS prediction/update is below this value because the residual also includes feature construction and shared pipeline overhead. |
| ClaimDecomp | 0.05 | 0.003 | Actual RLS prediction/update is below this value because the residual also includes feature construction and shared pipeline overhead. |

## B.3. Fixed Verifier Budget Selection Results

Table B.4 reports Macro-F1 for the three allocation strategies used in the fixed-budget selection analysis: generation-order top-$k$, random selection with seed 20, and surrogate-guided selection. The table includes the key budget points discussed in Chapter 5, plus low- and high-budget points to show the overall pattern. The two 50-trace settings use the same reported $k$ values so their trends can be compared directly.

**Table B.4:** Selected fixed-budget Macro-F1 values. The two 50-trace settings use the same reported $k$ values.

| Setting | Candidate pool | $k$ | Generation order | Random | Surrogate-guided |
|---|---|---|---|---|---|
| QuanTemp test | 15 traces | 3 | 46.42 | 47.06 | **47.14** |
| QuanTemp test | 15 traces | 5 | 48.34 | 48.85 | **49.27** |
| QuanTemp test | 15 traces | 7 | 48.97 | 49.10 | **50.06** |
| QuanTemp test | 15 traces | 10 | 49.80 | **50.29** | 50.18 |
| QuanTemp test | 15 traces | 13 | **50.34** | **50.34** | **50.34** |
| ClaimDecomp test | 50 traces | 5 | 42.81 | 42.49 | **43.51** |
| ClaimDecomp test | 50 traces | 10 | 42.81 | 44.94 | **45.57** |
| ClaimDecomp test | 50 traces | 15 | 43.68 | **45.38** | 43.62 |
| ClaimDecomp test | 50 traces | 20 | 41.90 | **44.23** | 43.21 |
| ClaimDecomp test | 50 traces | 25 | 42.72 | **42.88** | 41.85 |
| ClaimDecomp test | 50 traces | 30 | 42.21 | **42.83** | 41.80 |
| ClaimDecomp test | 50 traces | 40 | 42.78 | **43.31** | 42.35 |
| ClaimDecomp test | 50 traces | 45 | 41.72 | **42.81** | 42.35 |
| QuanTemp validation | 50 traces | 5 | **48.81** | 43.88 | 47.86 |
| QuanTemp validation | 50 traces | 10 | 49.73 | 45.36 | **50.68** |
| QuanTemp validation | 50 traces | 15 | **49.76** | 46.81 | 49.47 |
| QuanTemp validation | 50 traces | 20 | 48.12 | 48.62 | **48.76** |
| QuanTemp validation | 50 traces | 25 | 48.47 | 48.37 | **49.31** |
| QuanTemp validation | 50 traces | 30 | **49.03** | 48.55 | 48.49 |
| QuanTemp validation | 50 traces | 40 | 49.74 | 48.28 | **50.18** |
| QuanTemp validation | 50 traces | 45 | 49.50 | **50.41** | 49.95 |

Figure B.1 shows the same fixed-budget comparison with an additional verifier-score top-$k$ reference. This reference uses verifier scores from the already generated candidate pool to choose the top-$k$ traces, so it is a diagnostic upper bound on verifier-score information rather than a deployable budgeted method. It is also not guaranteed to be the highest Macro-F1 curve, because selecting by verifier score is not the same as selecting by gold-label correctness. On QuanTemp, surrogate-guided selection closes much of the gap to this reference at low and middle verifier budgets. On ClaimDecomp, the curves are noisier, which supports the main-text interpretation that the surrogate improves allocation in some budget regimes but does not uniformly dominate the simpler baselines.

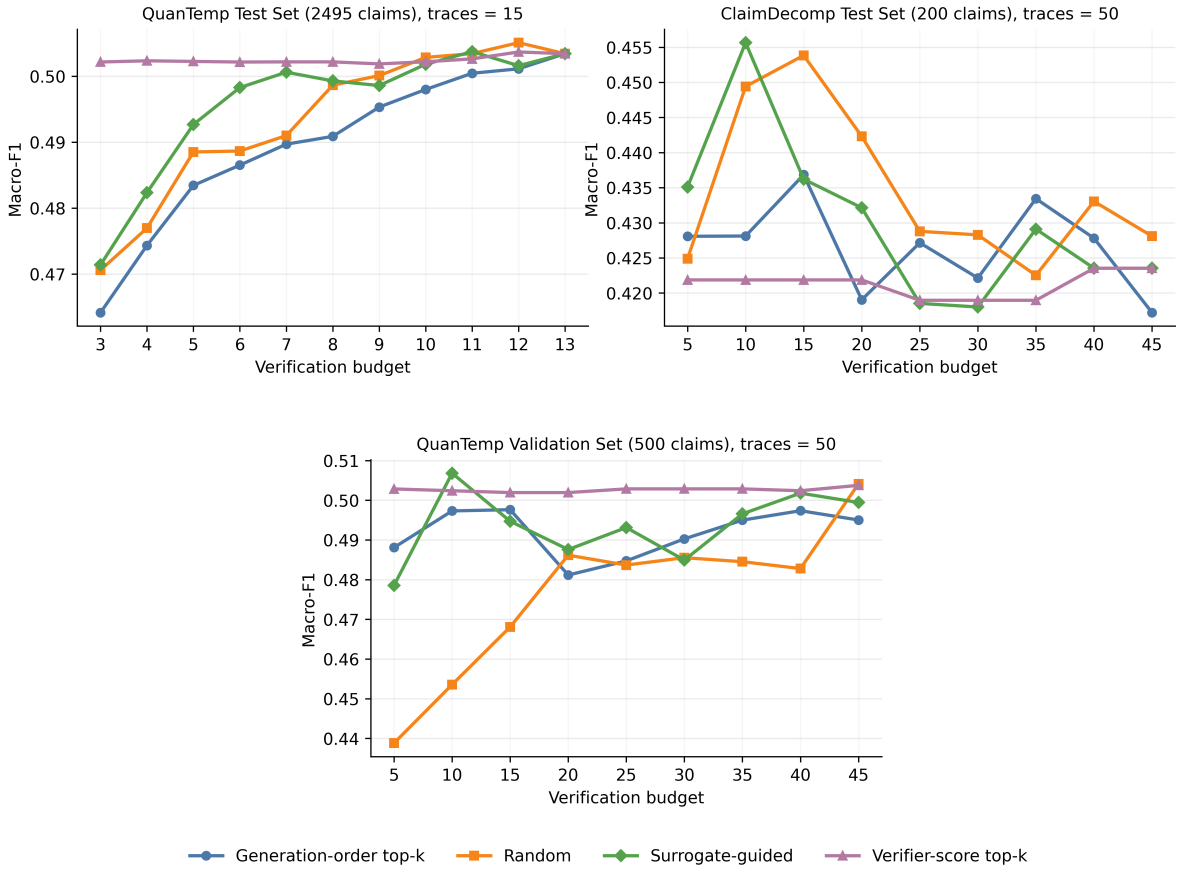



**Figure B.1:** Fixed-budget Macro-F1 with verifier-score top-$k$ upper-bound reference. The verifier-score top-$k$ curve uses verifier scores from the candidate pool and is included only as a diagnostic reference.

# C

# Qualitative Case Studies

This appendix gives representative qualitative examples supporting the analysis of Verifier-Guided Adaptive Stopping in Chapter 5. The examples show how evidence quality, verifier-score margins, and ambiguity in the retrieved evidence affect whether Adaptive Stopping terminates early or uses the full trace budget. These examples are illustrative and are not used as aggregate evidence.

## C.1. Verifier-Guided Adaptive Stopping Examples

**Type 1 – Unambiguous Claim: Stops at 3 traces (margin = 0.755)**

**Claim:** After a Columbus police officer fatally shot Ma'Khia Bryant, 16, in April 2021, LeBron James tweeted the officer was wrong to use deadly force.

**Retrieved Evidence:**

- " Ma'Khia Bryant was shot and killed by officer Nicholas Reardon in April 2021."
- "Those dreams were cut short after a Columbus police officer fatally shot Ms. Bryant on Tuesday afternoon, just moments after arriving at a chaotic disturbance outside her foster home."

*Gold:* supports *Pred:* supports ✓ *Best verifier score:* 0.998 *Traces used:* 3 / 15

**Why it stops early:** The evidence directly names the same person, event, and date as the claim. One trace is sufficient to produce a confident, well-grounded reasoning path (score 0.998), creating a large margin over the next label and triggering early stopping.

**Type 2 –Ambiguous Claim: Exhausts all 15 traces (max score = 0.985, margin = 0.016)**

**Claim:** "We are looking at the USMCA, NAFTA 2.0 trade deal. That would be very important and would add a half a point of GDP and 180,000 new jobs per year if we get that through."

**Retrieved Evidence:**

- "According to the U.S. International Trade Commission, USMCA will increase real

GDP by $68.2 billion and create 176,000 new American jobs."

- "These are substantial improvements from NAFTA. They represent benefits and new opportunities for Americans across the board."
- "The USMCA … will have a positive and overarching impact on the economy. The biggest impact … will come from new rules on international data transfers and e-commerce."

*Gold:* conflicting *Pred:* conflicting ✓ *Best verifier score:* 0.985 *Margin:* 0.016 *Traces used:* 15 / 15 | supports: 9 traces ≈ 0.87–0.97 conflicting: 6 traces ≈ 0.76–0.985

**Why it exhausts the budget:** The evidence partially supports the claim: 176,000 jobs is close to the claimed 180,000, and the GDP impact is positive. However, $68.2 billion absolute GDP gain does not directly correspond to "half a point" of GDP growth, and the "per year" framing in the claim is not confirmed by the evidence. Both supports and conflicting reasoning paths receive high verifier scores (0.87-0.985) across all 15 traces, and the margin between them never widens beyond 0.016, preventing early stopping.