LogiDebrief: A Signal-Temporal Logic Based Automated Debriefing Approach with Large Language Models Integration

Zirong Chen[1] , **Ziyan An**[1] , **Jennifer Reynolds**[2] , **Kristin Mullen**[2] , **Stephen Martini**[2] , **Meiyi Ma**[1]
[1]Department of Computer Science, Vanderbilt University, Nashville, Tennessee 37235, USA
[2]Metro Nashville Department of Emergency Communications, Nashville, Tennessee 37211, USA
{zirong.chen, ziyan.an, meiyi.ma}@vanderbilt.edu
{jennifer.reynolds, kristin.mullen, stephen.martini}@nashville.gov

Abstract

Emergency response services are critical to public safety, with 9-1-1 call-takers playing a key role in ensuring timely and effective emergency operations. To ensure call-taking performance consistency, quality assurance is implemented to evaluate and refine call-takers' skillsets. However, traditional human-led evaluations struggle with high call volumes, leading to low coverage and delayed assessments. We introduce *LogiDebrief*[1], an AI-driven framework that automates traditional 9-1-1 call debriefing by integrating Signal-Temporal Logic (STL) with Large Language Models (LLMs) for fully-covered rigorous performance evaluation. LogiDebrief formalizes call-taking requirements as logical specifications, enabling systematic assessment of 9-1-1 calls against procedural guidelines. It employs a three-step verification process: (1) contextual understanding to identify responder types, incident classifications, and critical conditions; (2) STL-based runtime checking with LLM integration to ensure compliance; and (3) automated aggregation of results into quality assurance reports. Beyond its technical contributions, LogiDebrief has demonstrated real-world impact. Successfully deployed at Metro Nashville Department of Emergency Communications, it has assisted in debriefing 1,701 real-world calls, saving 311.85 hours of active engagement. Empirical evaluation with real-world data confirms its accuracy, while a case study and extensive user study highlight its effectiveness in enhancing call-taking performance.

1 Introduction

Emergency response services are vital to public safety, with 9-1-1 call-takers as the first point of contact in crises, directly influencing response times and life-saving outcomes. Given their critical role, maintaining high performance is essential. To ensure consistency, emergency communication centers implement quality assurance programs that evaluate call-taker performance, enforce protocols, and provide actionable feedback. These programs use call reviews, guidecard cross-referencing, and protocol verification to enhance efficiency and emergency response effectiveness.

Despite their critical role, quality assurance programs across the U.S. face challenges in providing timely feedback due to high call volumes and limited resources [Ma *et al.*, 2018]. For example, during peak periods in 2024, the NYC Fire Department handled up to 6,500 emergency calls daily [NY, 2025], straining quality assurance personnel. As urban populations grow and emergency call volumes rise, these challenges intensify [Ma *et al.*, 2019; Ma *et al.*, 2020a]. Funding shortages and staffing constraints further hinder timely quality reviews [Afonso, 2021]. Timely feedback is essential for effective quality assurance [Adarkwah, 2021]. Delays reduce relevance, making it harder for call-takers to recall key details, address performance gaps, and reinforce best practices. Without prompt debriefing, quality assurance programs risk becoming bottlenecks, delaying critical insights needed to improve call-taker training and emergency response. If unaddressed, these challenges may compromise 9-1-1 call centers' ability to maintain high-performance standards, ultimately affecting response times and life-saving interventions.

Given these demands, an automated system is urgently needed for effective emergency call debriefing. While LLMs have advanced natural language processing [Rouzegar and Makrehchi, 2024], their application in this domain presents significant challenges. Our preliminary trials identify three key **challenges**: (1) *Step-by-step reasoning* is crucial for evaluating call-taker performance, as emergency calls require strict procedural adherence. LLMs must not only understand context but also apply structured reasoning. While In-Context Learning (ICL) techniques, such as Chain-of-Thought (CoT) prompting [Wei *et al.*, 2022], improve reasoning, studies show that LLMs still struggle with complex, high-stakes decision-making [Miao *et al.*, 2023; Huang *et al.*, 2023; Kambhampati, 2024]. Even advanced automatic reasoning methods [Zelikman *et al.*, 2022] exhibit weaknesses in scenarios requiring rigorous procedural verification [Wu *et al.*, 2024; McCoy *et al.*, 2024]. (2) *Cross-document retrieval and reference* remains another challenge. Call-takers rely on many complicated procedural documents, including guidecards, policies, and emergency protocols. While Retrieval-Augmented Generation (RAG) [Lewis *et al.*, 2020] assists

[1] More details: https://meiyima.github.io/angie.html

by fetching external documents, its accuracy is inconsistent, often retrieving outdated or irrelevant information and struggling with multi-document synthesis [Shi *et al.*, 2023; Shuster *et al.*, 2022]. Retrieval failures can result in missing critical protocols, significantly impacting evaluation. (3) The *complex nature of emergency calls* further complicates debriefing, as a single call may span multiple protocols (e.g., a motor vehicle accident may begin under police protocols but escalate to medical and fire due to injuries or hazards). Some calls exceed 20 minutes, making evaluation even more challenging. Combining ICL and RAG often results in excessively long prompts that exceed optimal context windows. Empirical studies [Weng *et al.*, 2024; Dong *et al.*, 2024; An *et al.*, 2024; Kuratov *et al.*, 2024] show that longer prompts degrade performance, leading to incomplete reasoning, ignored context, and lower factual consistency, as illustrated in Figure 1.

In this paper, we introduce *LogiDebrief*, the first framework, to our knowledge, designed to automatically and effectively assist in 9-1-1 call-taking debriefing. LogiDebrief integrates logic-enhanced reasoning with LLMs' language understanding, providing an effective approach to evaluating call-taker performance. Unlike traditional ICL methods that rely on lengthy prompts, LogiDebrief first collaborates with domain experts to decompose call-taking requirements into signal-temporal logic (STL) specifications [Maler and Nickovic, 2004]. During runtime checking, LLMs function as independent evaluators within STL to verify compliance. Once verification is complete, LogiDebrief aggregates results, generates quality assurance forms, and delivers actionable feedback with tailored explanations. This automated, just-in-time debriefing process enhances call-taker training and improves emergency response effectiveness.

Our **technical innovations** and **contributions** are: (1) We introduce *LogiDebrief*, a novel framework that automates 9-1-1 call-taking debriefing by integrating rigorous logic-based verification with LLM-powered analysis. (2) We decompose and formalize call-taking manuals into logic specifications through expert collaboration, ensuring standardized procedural verification, and improving consistency and reliability in 9-1-1 call debriefing. (3) We design an STL-integrated framework that seamlessly integrates LLMs as modular functions, enhancing procedural compliance while reducing the reliance on complex and lengthy prompts. (4) We empirically evaluate LogiDebrief's performance through extensive experiments with real-world data, demonstrating its effectiveness in delivering accurate and reliable debriefing assessments. (5) We conduct a real-world case study and a user study under practical deployment. The findings confirm that LogiDebrief is an effective tool for improving call-taking performance and training in emergency response settings.

Beyond its technical advancements, LogiDebrief delivers significant **social impact**: (1) Developed in collaboration with researchers and governmental agencies, LogiDebrief has been successfully deployed at Metro Nashville Department of Emergency Communications (MNDEC). It is now integrated into training programs for both active call-takers and trainees. (2) To date, it has assisted in debriefing 1,701 real-world calls, saving an estimated 311.85 working hours. (3) It facilitates debriefing for more than 200 call scenarios, covering various responder departments, call types, and life-threatening situations. (4) LogiDebrief has the potential to scale nationwide, offering automated debriefing solutions to emergency communication centers, particularly those operating in resource-constrained environments.

2 Motivating Study

Through discussions with MNDEC and a manual review of 1,244 past calls and debriefing results, we identified critical limitations in current 9-1-1 call center practices.

Insufficient Call Review Coverage and Delayed Feedback Emergency dispatch centers manage overwhelming call volumes daily, making comprehensive quality reviews increasingly difficult. At the local level, only 3.32% of calls undergo manual review, covering approximately 2,000 to 2,300 calls per month. Nationwide, review rates often remain below 10% due to resource constraints [Ma *et al.*, 2019]. Debriefing a single call takes an average of 11.5 minutes, leading to backlogs that delay actionable insights and further strain quality assurance personnel.

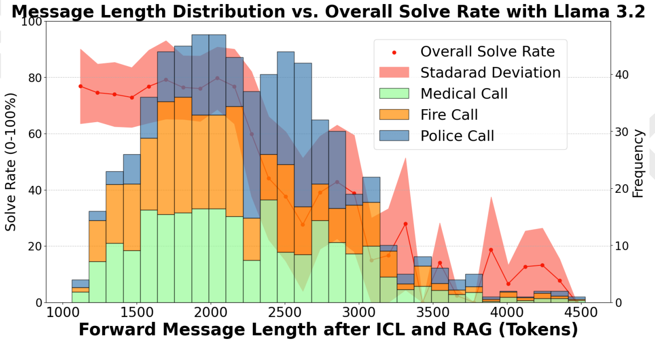


Figure 1: Message Length vs. One-shot Solve Rate with Llama 3.2. This figure shows the relationship between prompt length (in tokens) and solve rate, segmented by primary call categories (Medical, Fire, and Police). Shaded regions indicate standard deviations.

Challenges in LLM-Based Debriefing Workflows We evaluated debriefing workflows using real-world samples with recent LLMs, including Llama 3.2 [Meta, 2024], integrating ICL and RAG. Forwarded messages included step-by-step instructions, learning examples, and vectorized call-taking requirements. Our analysis compared LLM-generated debriefing results to ground truth while assessing performance relative to message length. As shown in Figure 1, calls typically generate 1,800 to 3,000 tokens, where performance drops approximately from 78% to 42%, with long-tail cases falling to 10%. These results underscore the challenge of maintaining accuracy as call scenarios grow more intricate and involve broader procedural references and checks, as they are also consistent with [Weng *et al.*, 2024; Dong *et al.*, 2024; An *et al.*, 2024; Kuratov *et al.*, 2024].

3 Problem Formulation and System Overview

Call debriefing evaluates emergency call-handling performance by reviewing past calls for protocol adherence, ensuring compliance, identifying improvements, and enhancing emergency response. A robust framework must account for conversational dynamics, filter relevant requirements, and assess procedural adherence objectively. LogiDebrief automates this by interpreting call context, filtering irrelevant requirements, verifying compliance, and aggregating results into a quality assurance form with clear, standardized feedback. Following this, we formulate the 9-1-1 call debriefing problem. A **9-1-1 call** is a structured dialogue between call-taker (a) and caller (b), represented as: $\omega_{(ab)} := \langle a_1, b_1, a_2, b_2, \ldots, a_t, b_t \rangle$. Where t is the number of the conversational turns, the call-taker's utterances and caller's utterances are defined correspondingly as, $\omega_{(a)} := \langle a_1, \ldots, a_t \rangle$ and $\omega_{(b)} := \langle b_1, \ldots, b_t \rangle$. 9-1-1 call-taking documents contain **requirements** $\mathcal{R} = \{r_1, r_2, \ldots, r_p\}$. that call-takers must meet while handling an emergency call. Any $r_i \in \{\top, \bot\}$ is a predicate. r_i is associated with a set of **preconditions** $\{\mathcal{P}_i \mid r_i\}$. Only when the entire $\mathcal{P}_i$ holds given a conversational signal ω, formally, $\mathbb{I}(\mathcal{P}_i \mid r_i)$, will the corresponding r_i be applied for checks. The **quality assurance forms** Ψ is based on the responder departments required for an emergency call, including fire, police, and medical. Each Ψ consists of multiple checks $\Psi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$. And $\forall \varphi \in \Psi, \varphi \in \{\text{Yes}, \text{No}, \text{(Caller) Refused}, \text{NA}\}$. Any φ is aggregated from multiple associated requirements, formally written as $\varphi = \mathcal{F}(\mathcal{R})$, where $\mathcal{F}$ is the aggregation function, and $\mathcal{R} = \{r_1, r_2, \ldots\}$ with each r_i inferred from its precondition $\mathcal{P}_i$.

4 Methodology

This section outlines LogiDebrief's workflow for 9-1-1 call debriefing. It systematically analyzes past calls against procedural guidelines, integrating formalized call-taking requirements (Section 4.1) with runtime monitoring (Section 4.2) to ensure compliance and identify gaps. LogiDebrief follows a 3-step runtime checking process: (1) Establishing context by identifying responders, call types, and critical situations while filtering inapplicable checks (Section 4.2.1). (2) Conducting runtime verification through logic-based rules to assess procedural adherence (Section 4.2.2). (3) Aggregating results into a quality assurance form, highlighting compliance, deviations, and actionable feedback for training (Section 4.2.3).

4.1 Formalizing Call-taking Requirements

Through discussions with the quality assurance team at MNDEC, we systematically reviewed and reconstructed existing call-taking requirements with domain expertise. By analyzing each requirement, we identified and formalized 2,215 distinct requirements, each specifying its preconditions. These span 57 general 9-1-1 call types, including heart problems, drowning, structure fires, and burglary, as well as six critical life-threatening protocols: airway control, Automated External Defibrillator (AED) usage, bleeding control, Cardiopulmonary Resuscitation (CPR), childbirth, and obstructed airways. Requirements were decomposed into STL specifications with preconditions using existing translation tools [Chen *et al.*, 2022a; Chen *et al.*, 2023; Chen *et al.*, 2022b]. For instance, in an animal bite case, if and only if the patient was bitten by a snake, the call-taker should instruct the caller *not* to elevate the extremity; otherwise, elevation is advised. This is formally expressed as $\Diamond_{[0,T]}$ "call-taker should warn caller not to elevate the extremity," with precondition $p_1 \wedge p_2$, where p_1 represents "the caller reporting an animal bite," and p_2 denotes "the patient was bitten by a snake."

4.2 STL-Based Runtime Monitoring with LLMs

We disentangle lengthy procedural explanations and retrieval augmentations in prompts by embedding ***independent modularized*** LLM calls as functions into STL. This approach extends STL's rigor, interpretability, and effectiveness [An *et al.*, 2025; Ma *et al.*, 2020a; Ma *et al.*, 2020b; Ma *et al.*, 2021] while reducing prompt complexity. Runtime examples are in Table 1. These functions operate over the conversational signal ω, enabling dynamic reasoning for assessing call-taking compliance. The debriefing process follows three key steps: (1) understanding the context, (2) runtime checking, and (3) aggregating results.

Step 1: Understanding the Context

Understanding the context of a 9-1-1 call is crucial for procedural adherence. Each emergency requires a structured assessment, including identifying responder departments, classifying the incident type, and recognizing life-threatening situations requiring immediate intervention. These factors shape the call-taker's approach, determining the sequence of questions, instructions, and protocols.

Determining the Responders The first step in contextualizing a call is *identifying the required responders*, denoted as $\hat{\mathrm{R}}$. Emergency services, fire, police, and medical, are analyzed independently for relevant indicators. We define the `SCENE` function, which returns $\top$ if a relevant scene description is detected and $\bot$ otherwise:

$$\texttt{SCENE}(\omega, \text{responders}) := \Diamond_{[0,T]} \underbrace{\big(\omega(t) \models \text{responders}\big)}_{\text{LLM Prompts}} \quad (1)$$

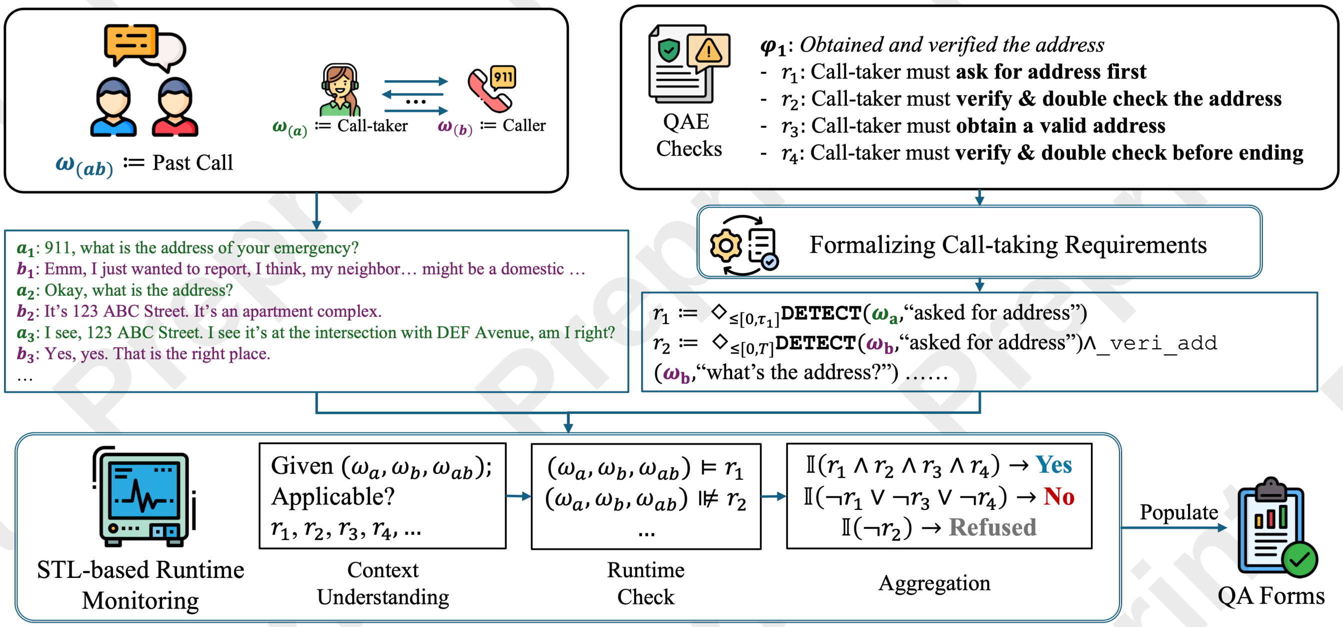
Iterating through the three categories, the system obtains all applicable responders:

$$\forall \mathrm{r} \in \{\text{fire}, \text{police}, \text{medical}\}, \quad \texttt{SCENE}(\omega, \mathrm{r}) \rightarrow \hat{\mathrm{R}} \cup \{\mathrm{r}\} \quad (2)$$

The final set $\hat{\mathrm{R}}$ includes all departments where the function evaluates to $\top$, ensuring accurate identification of required emergency services.

Identifying the Call Types After determining the required responders, the next step is to *classify the call type*, such as structure fires or heart problems, denoted as $\hat{\mathrm{T}}$. Instead of evaluating all incident types indiscriminately, classification is conditioned on the identified responders. We define the `TYPE` function, which classifies the call by analyzing its semantic context. It returns $\top$ if sufficient evidence supports the specified call type and $\bot$ otherwise:

$$\texttt{TYPE}(\omega, \text{type}) := \Diamond_{[0,T]} \underbrace{\big(\omega(t) \models \text{type}\big)}_{\text{LLM Prompts}} \quad (3)$$


$\omega_{(ab)}$ ≔ Past Call
$\omega_{(a)}$ ≔ Call-taker
$\omega_{(b)}$ ≔ Caller
911
QAE Checks
Formalizing Call-taking Requirements
STL-based Runtime Monitoring
Context Understanding
Runtime Check
Aggregation
Populate
QA Forms

φ_1: *Obtained and verified the address*
- r_1: Call-taker must **ask for address first**
- r_2: Call-taker must **verify & double check the address**
- r_3: Call-taker must **obtain a valid address**
- r_4: Call-taker must **verify & double check before ending**

a_1: 911, what is the address of your emergency?
b_1: Emm, I just wanted to report, I think, my neighbor… might be a domestic …
a_2: Okay, what is the address?
b_2: It's 123 ABC Street. It's an apartment complex.
a_3: I see, 123 ABC Street. I see it's at the intersection with DEF Avenue, am I right?
b_3: Yes, yes. That is the right place.
…

r_1 ≔ $\Diamond_{\leq[0,\tau_1]}$**DETECT**($\boldsymbol{\omega_a}$,"asked for address")
r_2 ≔ $\Diamond_{\leq[0,T]}$**DETECT**($\boldsymbol{\omega_b}$,"asked for address")∧`_veri_add`
($\boldsymbol{\omega_b}$,"what's the address?") ……

Given $(\omega_a, \omega_b, \omega_{ab})$;
Applicable?
$r_1, r_2, r_3, r_4, \ldots$

$(\omega_a, \omega_b, \omega_{ab}) \models r_1$
$(\omega_a, \omega_b, \omega_{ab}) \not\models r_2$
…

$\mathbb{I}(r_1 \land r_2 \land r_3 \land r_4) \rightarrow$ **Yes**
$\mathbb{I}(\neg r_1 \lor \neg r_3 \lor \neg r_4) \rightarrow$ **No**
$\mathbb{I}(\neg r_2) \rightarrow$ **Refused**

Figure 2: Overview of the LogiDebrief Workflow. It evaluates call-taker performance by analyzing past calls against formalized requirements. It extracts conversational signals $\omega_{(a)}$ and $\omega_{(b)}$ from past calls, then applies quality assurance evaluation (QAE) checks. During runtime monitoring with LLMs, it finalizes applicable checks based on call context, checks the compliance of each check, and aggregates results into a quality assurance form with actionable template-based feedback.

The system iterates over call types relevant to the confirmed responders:

$$\forall \mathrm{t} \in \text{Call Types} \mid \hat{\mathrm{R}}, \quad \texttt{TYPE}(\omega, \mathrm{t}) \rightarrow \hat{\mathrm{T}} \cup \{\mathrm{t}\}. \tag{4}$$

Thus, only incident types relevant to the identified responder departments are considered. E.g., if only police response is required, fire- or medical-related incidents such as structure fires or diabetic emergencies are excluded. The final result includes all incident types where the function evaluates to $\top$, ensuring accurate classification.

Alerting Critical Situations Following up, LogiDebrief identifies predefined *life-threatening situations* requiring immediate intervention. Each of the 6 critical conditions, airway control, AED, bleeding control, CPR, childbirth, and obstructed airways, is analyzed independently. We define the `CRITICAL` function to check if any of these conditions apply to a given call ω:

$$\texttt{CRITICAL}(\omega, \text{flag}) \coloneqq \Diamond_{[0,T]} \underbrace{(\omega(t) \models \text{flag})}_{\text{LLM Prompts}} \tag{5}$$

This process is formally expressed as:

$$\forall \mathrm{c} \in \text{Criticals}_{\times 6}, \quad \texttt{CRITICAL}(\omega, \mathrm{c}) \rightarrow \hat{\mathrm{C}} \cup \{\mathrm{c}\} \tag{6}$$

where $\hat{\mathrm{C}}$ represents the set of flagged critical situations. LogiDebrief iterates through all six conditions, ensuring that any applicable life-threatening scenario is detected and appropriate emergency protocols are triggered without delay.

Finalizing Checks After determining $\hat{\mathrm{R}}$, $\hat{\mathrm{T}}$, and $\hat{\mathrm{C}}$, LogiDebrief finalizes checks. These checks dynamically adjust based on scene information denoted as $\Gamma(.)$; e.g., medical-related forms verify patient assessment, while police-related forms ensure scene safety.

$$\Psi = \Gamma(\hat{\mathrm{R}}), \quad \Psi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\} \tag{7}$$

While scene information defines the form's structure, refinements based on $\hat{\mathrm{T}}$ and $\hat{\mathrm{C}}$ update specific requirements without introducing new structural checks. Each check φ in Ψ is linked to a set of requirements $\{r_1, r_2, \ldots\}$, which adapt based on the emergency scenario:

$$\forall \varphi \in \Psi, \quad \varphi \leftarrow \varphi \oplus \Delta(\hat{\mathrm{T}}, \hat{\mathrm{C}}) \tag{8}$$

where $\oplus$ updates requirements using relevant call-taking manuals while preserving form structure. $\Delta(\hat{\mathrm{T}}, \hat{\mathrm{C}})$ applies context-specific refinements; e.g., a cardiac arrest call updates breathing assessments to explicitly verify chest compressions. After applying $\oplus\Delta(\hat{\mathrm{T}}, \hat{\mathrm{C}})$ to each $\varphi \in \Psi$, the final quality assurance form Ψ is obtained.

After obtaining Ψ, we iterate through each $\varphi \in \Psi$ to check compliance. For each φ, we retrieve its associated requirements $\{r_1, r_2, \ldots, r_i\}$ and corresponding preconditions $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_i\}$. To facilitate this process, we define the `SCAN` function, which verifies whether a precondition holds in ω. It returns $\top$ if the condition is met at any time t within the observation window, and $\bot$ otherwise:

$$\texttt{SCAN}(\omega, \text{precondition}) \coloneqq \Diamond_{[0,T]} \underbrace{(\omega(t) \models \text{precondition})}_{\text{LLM Prompts}} \tag{9}$$

The overall precondition evaluation is represented as:

$$\forall p \in \{\mathcal{P} \mid r, \forall r \in \varphi\}, \quad \texttt{SCAN}(\omega, p) \tag{10}$$

$\varphi_{\textbf{address}}$: **Address Check**	
r_1: call-taker asked for address in first τ_1 turns.	$\Diamond_{\leq[0,\tau_1]}\texttt{DETECT}(\omega_a, \text{‘ask address’})$
r_2: caller provided a valid address.	$\Diamond_{[0,T]}\big(\texttt{DETECT}(\omega_b, \text{‘provide address’})\wedge$ $\texttt{veri\_add}(\texttt{answer}(\omega_b, \text{‘what’s the address?’}))\big)$
r_3: call-taker verified the obtained address with nearby geo-info.	$\Box_{[0,T]}\big(\texttt{DETECT}(\omega_b, \text{‘provide address’}) \wedge \texttt{veri\_add}(\texttt{answer}(\omega_b, \text{‘what’s the address?’})) \rightarrow$ $\Diamond_{\leq\tau}\big(\texttt{DETECT}(\omega_a, \text{‘double checks address’})\wedge$ $\texttt{veri\_add}(\texttt{answer}(\omega_a, \text{‘what’s address?’}), \texttt{answer}(\omega_b, \text{‘what’s address?’}))\big)\big)$
r_4: call-taker verified the address again before getting disconnected.	$\Diamond_{[T-\tau_2,T]}\big(\texttt{DETECT}(\omega_a, \text{‘double checks address’})\wedge$ $\texttt{veri\_add}(\texttt{answer}(\omega_a, \text{‘what’s address?’}))$
$\varphi_{\textbf{name}}$: **Caller Name Check** & $\varphi_{\textbf{phone}}$: **Caller Phone Check**	
r_1: call-taker asked for both first and last name / phone number.	$\Diamond_{[0,T]}\big(\texttt{DETECT}(\omega_a, \text{‘ask for full name / phone number’})$
r_2: caller provided both first and last name / phone number.	$\Diamond_{[0,T]}\big(\texttt{DETECT}(\omega_b, \text{‘provide full name / phone number’})$
r_3: call-taker followed up with caller’s name / phone number.	$\Box_{[0,T]}\big(\texttt{DETECT}(\omega_b, \text{‘provides name / phone’}) \rightarrow$ $\Diamond_{\leq\tau_1}\texttt{DETECT}(\omega_a, \text{‘follows up on name / phone’})\big)$
$\varphi_i \in \Psi$: **Conditional Checks**	
r_1: if the scene is potentially not safe for police officers, call-taker obtained scene safety info.	$\Diamond_{\leq[0,\tau_3]}\texttt{DETECT}(\omega_{ab}, \text{‘scene safety info obtained’})$
r_2: if the patient is an infant and not breathing, call-taker should do [*infant CPR*: *step 1, step 2, ...*].	$\Diamond_{\leq[0,\tau_4]}\texttt{DETECT}(\omega_a, \text{‘call-taker instructs [}\textit{infant CPR}\text{: }\textit{step 1, step 2, ...}\text{]’})$
r_3: if the call involves medical emergency, call-taker should check if patient is breathing normally.	$\Diamond_{\leq[0,\tau_5]}\texttt{DETECT}(\omega_a, \text{‘checked patient breathing’})$
r_4: if there is any suspicious vehicle spotted on the scene, call-taker should ask for detailed vehicle descriptions.	$\Diamond_{\leq[0,\tau_6]}\texttt{DETECT}(\omega_a, \text{‘asked for vehicle description’})$
r_5: if the roadway hazard is blocking the traffic, call-taker should warn caller not to move the hazard by themselves.	$(\Diamond_{\leq[0,\tau_7]}\texttt{DETECT}(\omega_a, \text{‘warn caller not to move the hazard’}))$
r_6: if the caller reports an odor, call-taker should warn caller to avoid using energized equipment that could cause a spark.	$\Diamond_{\leq[0,\tau_8]}\texttt{DETECT}(\omega_a, \text{‘warn caller not to use energized equipment’})$

Table 1: A runtime example of both conditional and unconditional checks in natural languages and STL specifications. Each of the conditional checks φ satisfies Equation 12. All τ are adaptable hyper-parameters for different call-taking requirements.

$$\mathbb{I}(\mathcal{P} \mid r) = \bigwedge_{p \in \{\mathcal{P}\mid r\}} \texttt{SCAN}(\omega, p) \tag{11}$$

If the preconditions do not hold, requirement $\mathcal{R}_i$ is skipped and excluded from runtime monitoring. Ultimately, a quality assurance form Ψ should satisfy:

$$\forall \varphi \in \Psi, \forall r \in \varphi \quad \mathbb{I}(\mathcal{P} \mid r) = \top \tag{12}$$

Step 2: Checking the Runtime

We define functions to verify requirements. The DETECT function checks whether a specific *action* occurs within the conversation signal ω, returning $\top$ if detected at any time t within the observation window and $\perp$ otherwise:

$$\texttt{DETECT}(\omega, \text{action}) := \Diamond_{[0,T]} \underbrace{\big(\omega(t) \models \text{action}\big)}_{\text{LLM Prompts}} \tag{13}$$

Additional non-STL functions further analyze ω:

$$\texttt{answer}(\omega, \text{query}) \rightarrow a. \tag{14}$$

The answer function integrates LLMs for question-answering, retrieving the most relevant response a from ω. If no answer is found, it returns an empty string.

$$\texttt{veri\_add}(\text{addresses}) \rightarrow \{\top, \perp\} \tag{15}$$

The veri_add function utilizes Google Geocoding and Places API to validate addresses. If given an empty string, it returns $\top$ by default. For a *single address*, it returns $\top$ if successfully located on a map; otherwise, $\perp$. For *two addresses*, it returns $\top$ if geographically close; otherwise, $\perp$. These functions are embedded in STL specifications for just-in-time runtime verification after each call. Examples of their integration are shown under each φ in Table 1.

Step 3: Aggregating the Results

After having satisfaction for each $r_i \in \mathcal{R}$, we aggregate the result to populate:

$$\varphi \leftarrow \mathcal{F}(\mathcal{R}), \quad \varphi \in \{\text{Yes}, \text{No}, \text{Refused}, \text{NA}\} \tag{16}$$

This aggregation step is instructed back to trainees with template-based natural language generation as explanations for the populated results, e.g., “Your overall evaluation at this check is NO, because you missed r.”

Unconditional Checks According to the call-taking manual, three checks, $\varphi_1, \varphi_2, \varphi_3$ in Table 1, are always examined regardless of call context: *address*, caller *name*, and phone *number*, each with distinct aggregation rules.

The address check categorizes verification performance into three outcomes: (1) *Yes*: The call-taker successfully requests (r_1), verifies (r_3), and reconfirms (r_4) the address, and the caller provides a valid one (r_2). For example, if a geocodable location is confirmed before call termination, the outcome is *Yes*. (2) *No*: The call-taker fails to ask for (r_1), verify (r_3), or reconfirm (r_4) the address, regardless of caller response. If geographical verification is neglected, the call is classified as *No*. (3) *Refused*: If the caller explicitly refuses to provide an address (r_2), the outcome is *Refused*, as the failure is beyond the call-taker’s control, e.g., a distressed caller declining to disclose their location. Formally:

$$\varphi_{\text{address}} = \begin{cases} \text{Yes}, & \text{if } r_1 \wedge r_2 \wedge r_3 \wedge r_4 \\ \text{No}, & \text{if } \neg r_1 \vee \neg r_3 \vee \neg r_4 \\ \text{Refused}, & \text{if } \neg r_2 \end{cases} \tag{17}$$

The outcomes of caller name and phone checks classify the call-taker’s performance in collecting caller identity into three categories: (1) **Yes**: The call-taker correctly requests (r_1), receives (r_2), and follows up on (r_3) the caller’s name and phone number, ensuring full compliance. For example, if all details are requested, provided, and confirmed, the outcome is *Yes*. (2) **No**: The call-taker fails to request (r_1) or follow up (r_3) on the information, regardless of whether the caller provides it. If verification is omitted, the call is classified as *No*. (3) **Refused**: The caller explicitly refuses to provide their name or phone number (r_2), making the failure beyond the call-taker's control, e.g., a caller declining to disclose their identity despite multiple requests. Formally:

$$\varphi_{\text{name}}, \varphi_{\text{phone}} = \begin{cases} \text{Yes}, & \text{if } r_1 \wedge r_2 \wedge r_3 \\ \text{No}, & \text{if } \neg r_1 \vee \neg r_3 \\ \text{Refused}, & \text{if } \neg r_2 \end{cases} \tag{18}$$

Conditional Checks The outcome of any conditional checks φ_i depends on the satisfaction of all monitored requirements r_i: (1) **Yes**: All applicable conditional checks are met, meaning every monitored r_i returns $\top$. For example, if the scene was unsafe (r_1) and the call-taker obtained scene safety information, the outcome is *Yes*. (2) **No**: At least one monitored requirement fails ($r_i = \bot$). For instance, if a medical emergency is detected (r_3), but the call-taker fails to check the patient's breathing, the outcome is *No*. (3) **NA**: No requirements were monitored due to unsatisfied preconditions.

$$\varphi_i = \begin{cases} \text{Yes}, & \text{if } \forall r \in \varphi_i, r = \top \\ \text{No}, & \text{if } \exists r \in \varphi_i, r = \bot \\ \text{NA}, & \text{if } \varphi_i = \emptyset \end{cases} \tag{19}$$

5 Evaluation

LogiDebrief is a pioneering AI-driven system designed to automate 9-1-1 call debriefing. Given its novelty, limited literature exists to guide its evaluation. To ensure a comprehensive assessment, we evaluate its effectiveness through both quantitative benchmarking and real-world case studies. In addition, we conducted a user study to further assess its impact on enhancing call-taking performance.

Quantitatively, we investigate ***how effectively LogiDebrief debriefs 9-1-1 calls***. We first evaluate LogiDebrief on 1,244 real-world calls with debriefing results provided by quality assurance experts at MNDEC. However, since professional 9-1-1 operators handle these calls, errors are rare, potentially leading to an inflated false positive rate. Additionally, this dataset lacks coverage of rare but critical incidents (e.g., aircraft crashes, nuclear leaks). To address these limitations, we *construct a diverse dataset* encompassing various call types and call-taker proficiency levels: (1) Defining all requirements with their preconditions (e.g., snake vs. non-snake bites); (2) Using LLMs to generate simulated 9-1-1 reports under role-play [Chen *et al.*, 2025]; and (3) Interacting with controlled actions [Chen *et al.*, 2024b] where call-takers access only a percentage (α) of requirements. Those scripted actions also generate the ground truth. Here, α represents familiarity level; higher α values indicate greater adherence to required actions. For instance, in an animal bite emergency, a scripted call-taker may fail to ask about the animal type if *"What type of animal caused the bite?"* is masked. We set $\alpha = 25, 50, 75$ to simulate varying proficiency levels, totaling 13,200 calls with corresponding quality assurance forms. Performance is reported with F-1 scores for $\{\text{Yes}, \text{No}, \text{Refused}, \text{NA}\}$ after multi-fold validation. Proprietary LLMs (GPT-4o, DeepSeek-v3-671B) and reasoning models (OpenAI-o1) are tested via API, while open-source and smaller LLMs run with 128 GB RAM, AMD Ryzen Threadripper Pro 7975WX, and NVIDIA RTX 6000 Ada.

Qualitatively, we focus on ***how effectively LogiDebrief enhances call-taking performance in real-world settings***. To assess this, we conduct a case study at MNDEC. Additionally, we conducted a user study to further validate LogiDebrief's effectiveness in enhancing call-taker training. See complete user study result in the Appendix [Chen and Ma, 2025].

5.1 Effectiveness in Call Debriefing

We evaluate LogiDebrief's performance following baseline setups: (1) *Vanilla LLMs*, where the full quality assurance form Ψ (Eq. 8) is provided as input, and responses are generated directly. (2) *LLMs with RAG*, utilizing vectorized call-taking manuals as knowledge bases without logical structuring. (3) *LLMs with RAG+ICL*, combining RAG with Chain-of-Thought reasoning and Few-Shot examples for procedural explanations. (4) *Reasoning frameworks* tested with necessary step-by-step instructions. We evaluate these setups using available LLM backends (Llama3.2-3B [Meta, 2024], Gemma2-9B [Google, 2024], DeepSeek-v3-671B [DeepSeek, 2024], and GPT-4o [OpenAI, 2024a]) and Reasoners (OpenAI-o1-2024-12-17 [OpenAI, 2024b] and DeepSeek-r1 [DeepSeek, 2025]).

Tab. 2 presents key insights across real-world and emulation scenarios: (1) *Vanilla LLMs underperform*, with F1 scores below 40% (e.g., Llama 3.2 Vanilla: 37.11±16.93 in unconditional checks). This confirms that call debriefing requires multi-step validation and logical reasoning, which standard LLMs struggle with. Even RAG, which incorporates call-taking manuals, provides minimal improvement (e.g., Llama 3.2 RAG: 40.99±13.72), as manuals contain static rules without reasoning mechanisms, limiting generalization beyond simple lookups. (2) *ICL+RAG improves step-by-step reasoning* and serves as the strongest LLM-based alternative for procedural verification. Thus, we consider it an approximation of LogiDebrief without STL. However, lacking explicit logical constraints, it remains prone to reasoning errors, particularly in strict procedural checks (e.g., GPT-4o ICL+RAG: 87.78±10.79). (3) *Reasoning models provide only marginal improvements*, with DeepSeek-r1 and OpenAI-o1 achieving 88.21±9.79 and 86.84±10.18, respectively. While they exhibit better problem comprehension, they lack procedural enforcement. LogiDebrief surpasses all baselines by integrating STL, ensuring rigorous protocol adherence beyond what LLMs alone can achieve. *Overall, LogiDebrief outperforms all baselines across real-world and emulation datasets, demonstrating that integrating formalized logic with LLM reasoning yields the most effective call debriefing performance.*

5.2 Case Study: LogiDebrief in the Field

We conducted a case study of LogiDebrief under its 4-week active engagement at MNDEC (Nov 2024 – Jan 2025) under daily use and 2 training sessions. In the study, LogiDebrief cross-reviewed 1,244 calls alongside human debriefing

			REAL-WORLD			EMULATION								
						α =25			α =50			α =75		
			$\forall\varphi \in \Psi$		Ψ	$\forall\varphi \in \Psi$		Ψ	$\forall\varphi \in \Psi$		Ψ	$\forall\varphi \in \Psi$		Ψ
			Unconditional	*Conditional*		*Unconditional*	*Conditional*		*Unconditional*	*Conditional*		*Unconditional*	*Conditional*	
Typical LLMs	Llama 3.2	Vanilla	37.11±16.93	36.24±16.10	7.12±5.03	29.51±8.01	31.69±4.36	8.97±7.57	25.12±9.23	32.23±10.72	7.90±6.47	35.62±17.17	32.50±9.82	8.85±7.03
		RAG	40.99±13.72	36.04±17.09	10.85±7.17	34.04±9.49	31.51±8.37	8.66±3.27	30.29±12.39	28.22±9.43	9.56±5.79	34.19±13.27	29.88±10.61	7.71±6.70
		ICL+RAG	75.39±14.33	78.24±13.33	26.85±15.89	71.73±8.94	74.12±7.53	20.60±14.14	81.17±9.15	77.61±11.11	22.06±14.14	73.13±18.29	78.09±11.64	24.11±12.87
	Gemma 2	Vanilla	37.81±17.15	27.56±12.45	11.62±7.49	36.94±9.20	31.10±11.05	9.44±3.00	34.89±13.94	39.38±9.76	11.08±10.82	41.43±15.11	45.61±11.78	11.18±6.55
		RAG	38.16±15.90	29.52±11.06	11.50±10.64	39.93±11.75	33.38±12.73	8.73±7.73	32.37±11.66	34.14±15.15	14.19±8.32	42.89±17.86	44.75±12.58	10.53±7.22
		ICL+RAG	71.88±15.94	73.91±14.55	30.97±11.22	72.05±9.95	73.91±13.81	25.18±14.14	72.61±15.03	73.69±10.12	25.81±11.58	69.82±10.23	68.43±12.64	23.18±9.57
	DeepSeek-v3	Vanilla	45.22±15.29	52.47±14.51	11.57±6.44	48.32±19.53	50.41±13.64	12.74±11.59	43.71±13.09	46.53±11.98	11.54±7.74	48.53±13.77	45.00±14.59	13.13±9.45
		RAG	59.75±11.39	57.49±18.66	19.11±8.34	53.64±16.84	58.24±17.61	12.68±8.89	56.29±11.46	57.97±18.12	13.40±7.38	57.74±13.76	56.27±12.11	15.97±8.96
		ICL+RAG	86.21±12.95	87.29±11.93	55.21±2.31	86.55±13.33	87.95±10.05	52.67±13.47	84.49±13.95	85.82±12.87	54.54±9.43	86.19±11.71	84.99±14.39	55.08±7.49
	GPT-4o	Vanilla	41.60±15.26	52.59±13.72	10.54±8.87	54.50±17.05	55.72±11.90	12.00±8.20	54.24±10.36	56.43±11.17	13.93±9.45	55.40±16.62	59.41±11.14	12.95±4.25
		RAG	54.51±18.10	58.16±17.81	17.23±7.03	56.67±13.98	61.83±14.50	21.54±10.52	62.84±10.49	59.52±12.19	15.49±6.72	56.57±15.54	57.15±12.14	16.93±6.81
		ICL+RAG	88.16±10.65	87.78±10.79	58.91±13.14	88.31±11.58	87.03±11.79	57.13±13.98	85.02±13.71	85.79±12.57	54.13±14.46	87.11±12.04	84.60±14.76	53.16±13.84
Reasoners	Deepseek-r1		86.84±10.18	87.75±7.58	60.50±12.44	88.88±10.62	87.76±11.83	60.20±14.48	86.57±13.19	86.23±11.67	58.91±12.10	86.09±11.33	85.24±12.44	55.22±14.80
	OpenAI-o1		88.21±9.79	89.83±8.17	63.74±10.11	87.63±12.37	88.75±11.25	59.62±15.13	87.50±12.50	89.33±10.22	60.93±17.83	88.45±11.55	87.63±10.37	58.52±16.11
LogiDebrief	Llama 3.2		81.30±4.02	88.62±7.63	58.80±6.40	87.90±3.13	88.78±6.33	54.33±9.05	85.52±5.02	87.22±7.11	50.12±6.46	84.51±3.21	87.96±6.33	51.52±5.76
	Gemma 2		77.75±4.16	80.78±6.12	59.24±2.35	78.29±4.88	79.70±4.49	52.40±6.08	80.45±4.10	81.11±5.00	51.49±2.71	78.15±3.61	80.98±3.52	52.24±8.14
	DeepSeek-v3		92.63±5.08	90.89±4.05	84.75±5.25	93.90±4.10	91.43±5.94	86.36±6.14	92.68±6.51	90.86±5.33	82.25±6.35	91.10±4.38	91.70±2.47	83.06±5.51
	GPT-4o		**95.93**±**4.07**	**94.40**±**5.60**	**94.33**±**5.67**	**94.39**±**5.61**	**95.07**±**4.93**	**94.84**±**5.16**	**95.45**±**4.55**	**95.38**±**4.62**	**94.62**±**5.38**	**94.49**±**5.51**	**95.61**±**4.39**	**94.04**±**5.96**

Table 2: Evaluation of LogiDebrief with REAL-WORLD and EMULATION data compared with baselines. α is call-taker 'proficiency levels': α percentage of the required actions are taken during the scripted emulation. At the φ level, performance is evaluated per check φ. At the Ψ level, a response is counted as correct only if the entire set is populated accurately. Performance is reported in multi-fold F-1 scores as %.

and independently analyzed 457 calls. A total of 29 participants contributed, including 16 trainees, 5 active call-takers, and 8 training/quality assurance officers, providing 37 feedback entries. We share the following findings: (1) ***Timeliness.*** Traditional quality assurance feedback is provided at the end of a shift, making it difficult for call-takers, who handle over 80 calls daily, to recall specific interactions. This delay reduces evaluation effectiveness and limits immediate skill reinforcement. LogiDebrief delivers just-in-time feedback, generating quality assurance reports in under 6 seconds per minute of call audio. Compared to the 11.5-minute manual review process, it reduces evaluation time to 4.45% (<30 seconds) per call while maintaining accuracy, saving over estimated 311 working hours. A quality assurance officer noted: *"The feedback was quick and spot-on. It even caught the mistakes I made on purpose. This can really save a lot of time."* (2) ***Higher Coverage.*** LogiDebrief boosted call review coverage by 73.96% to 85.05%, processing 1,701 more calls compared to previous 2,000 to 2,300 per 4 weeks under human efforts. (3) ***Comprehensiveness.*** Traditional quality assurance often emphasizes errors without reinforcing correct practices. Feedback can be generic, making it harder for call-takers to extract actionable insights. LogiDebrief provides balanced assessments, highlighting both strengths and areas for improvement. Its STL-enhanced check offers step-by-step guidance, clarifying why specific actions were correct or required adjustment. One call-taker shared: *"It walked me through step by step instead of just flagging mistakes, so I knew exactly what went wrong and how to fix it."* In summary, LogiDebrief enhances call-taking performance by providing timely, accurate, and actionable feedback. By reducing evaluation time, increasing review coverage, and improving instructional clarity, it supports continuous learning and strengthens procedural consistency in emergency response.

6 Related Work

Automated debriefing is well-studied in education and medical training, where structured feedback enhances skill development. Intelligent tutoring systems provide adaptive feedback for language learning, STEM education, and problem-solving but focus on static assessments rather than real-time procedural evaluation [Graesser *et al.*, 2012]. In medical training, AI-assisted tools assess procedural adherence in surgical simulations and emergency medicine [Toews *et al.*, 2021]. However, emergency call-taking remains largely overlooked despite its need for timely feedback. **Large Language Models for procedural checks** face reliability challenges. Chain-of-Thought prompting [Wei *et al.*, 2022] improves reasoning but does not ensure strict adherence, leading to hallucinations and missing steps [Turpin *et al.*, 2024; Ling *et al.*, 2024]. Retrieval-Augmented Generation (RAG) [Lewis *et al.*, 2020] improves factual accuracy but cannot guarantee retrieving relevant procedural guidelines, making it unreliable for high-stakes verification [Chen *et al.*, 2024a; Wang *et al.*, 2024]. Self-verification improves consistency but lacks procedural rigor [Zelikman *et al.*, 2022; Chung *et al.*, 2024], while longer prompts degrade multi-step procedural integration [Weng *et al.*, 2024]. More robust verification is needed. Extended related work is available in the Appendix [Chen and Ma, 2025].

7 Summary

In this paper, we introduce LogiDebrief, the first AI-driven framework for automating and assisting 9-1-1 call-taking debriefing. Integrating logic-driven procedural verification with LLM-powered analysis, LogiDebrief enables rigorous call-taker performance evaluation. Evaluation and case studies confirm its effectiveness in debriefing real-world 9-1-1 calls and enhancing call-taking performance.

This work can support emergency communication centers with limited resources by assisting with quality assurance and reducing manual debriefing burdens. With over 6,000 emergency communication centers across the US, it offers an effective approach for call-taker performance enhancement. Beyond emergency response, LogiDebrief's framework can potentially extend to structured compliance audits in other training spaces, such as medical triage and law enforcement.

Acknowledgments

This work was supported in part by the U.S. National Science Foundation under Grants 2427711 and 2443803, the Google Academic Research Award, OpenAI Researcher Access Program, and the U.S. Department of Education under Grant R305C240010. The opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsoring agencies.

Stakeholder Discussion *LogiDebrief* was co-designed and developed with Metro Nashville Department of Emergency Communications (MNDEC). Their domain expertise directly assisted in shaping the system's design, from surfacing critical quality assurance gaps to co-specifying over 2,200 procedural requirements in formal logic. Beyond problem framing, MNDEC participated in refining runtime thresholds, validating output interpretations, and integrating the tool into ongoing training. Beyond the reported 4-week case study, *LogiDebrief* continues to be used as part of MNDEC's ongoing training and quality assurance workflows. The partnership transitioned from initial prototyping to field deployment, establishing a scalable model for AI-assisted quality assurance that aligns with operational needs and is now being explored for broader regional adoption.

References

[Adarkwah, 2021] Michael Agyemang Adarkwah. The power of assessment feedback in teaching and learning: a narrative review and synthesis of the literature. *SN Social Sciences*, 1(3):75, 2021.

[Afonso, 2021] Whitney Afonso. Planning for the unknown: Local government strategies from the fiscal year 2021 budget season in response to the covid-19 pandemic. *State and Local Government Review*, 53(2):159–171, 2021.

[An *et al.*, 2024] Chenxin An, Jun Zhang, Ming Zhong, Lei Li, Shansan Gong, Yao Luo, Jingjing Xu, and Lingpeng Kong. Why does the effective context length of llms fall short? *arXiv preprint arXiv:2410.18745*, 2024.

[An *et al.*, 2025] Ziyan An, Xia Wang, Hendrik Baier, Zirong Chen, Abhishek Dubey, Taylor T. Johnson, Jonathan Sprinkle, Ayan Mukhopadhyay, and Meiyi Ma. Combining LLMs with logic-based framework to explain MCTS. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025)*, 2025.

[Chen and Ma, 2025] Zirong Chen and Meiyi Ma. Logidebrief: Appendix and supplementary materials. https://meiyima.github.io/angie.html, 2025. Accessed: 2025-05-07.

[Chen *et al.*, 2022a] Zirong Chen, Isaac Li, Haoxiang Zhang, Sarah Preum, John A Stankovic, and Meiyi Ma. Cityspec: An intelligent assistant system for requirement specification in smart cities. In *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 32–39. IEEE, 2022.

[Chen *et al.*, 2022b] Zirong Chen, Isaac Li, Haoxiang Zhang, Sarah Preurn, John A Stankovic, and Meiyi Ma. An intelligent assistant for converting city requirements to formal specification. In *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 174–176. IEEE, 2022.

[Chen *et al.*, 2023] Zirong Chen, Isaac Li, Haoxiang Zhang, Sarah Preum, John A Stankovic, and Meiyi Ma. Cityspec with shield: A secure intelligent assistant for requirement formalization. *Pervasive and Mobile Computing*, 92:101802, 2023.

[Chen *et al.*, 2024a] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17754–17762, 2024.

[Chen *et al.*, 2024b] Zirong Chen, Xutong Sun, Yuanhe Li, and Meiyi Ma. Auto311: A confidence-guided automated system for non-emergency calls. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 21967–21975, 2024.

[Chen *et al.*, 2025] Zirong Chen, Elizabeth Chason, Noah Mladenovski, Erin Wilson, Kristin Mullen, Stephen Martini, and Meiyi Ma. Sim911: Towards effective and equitable 9-1-1 dispatcher training with an llm-enabled simulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 27896–27904, 2025.

[Chung *et al.*, 2024] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

[DeepSeek, 2024] DeepSeek. Deepseek-v3: Scaling open large language models with moe, 2024.

[DeepSeek, 2025] DeepSeek. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[Dong *et al.*, 2024] Zican Dong, Junyi Li, Xin Men, Wayne Xin Zhao, Bingbing Wang, Zhen Tian, Weipeng Chen, and Ji-Rong Wen. Exploring context window of large language models via decomposed positional vectors. *arXiv preprint arXiv:2405.18009*, 2024.

[Google, 2024] Google. Gemma 2: Improving open language models at a practical size, 2024.

[Graesser *et al.*, 2012] Arthur C Graesser, Mark W Conley, and Andrew Olney. Intelligent tutoring systems. *American Psychological Association*, 2012.

[Huang *et al.*, 2023] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

[Kambhampati, 2024] Subbarao Kambhampati. Can large language models reason and plan? *Annals of the New York Academy of Sciences*, 1534(1):15–18, 2024.

[Kuratov *et al.*, 2024] Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *arXiv preprint arXiv:2406.10149*, 2024.

[Lewis *et al.*, 2020] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[Ling *et al.*, 2024] Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic, and Hao Su. Deductive verification of chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, 36, 2024.

[Ma *et al.*, 2018] Meiyi Ma, John A. Stankovic, and Lu Feng. Cityresolver: A decision support system for conflict resolution in smart cities. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 55–64, 2018.

[Ma *et al.*, 2019] Meiyi Ma, Sarah M Preum, Mohsin Y Ahmed, William Tärneberg, Abdeltawab Hendawi, and John A Stankovic. Data sets, modeling, and decision making in smart cities: A survey. *ACM Transactions on Cyber-Physical Systems*, 4(2):1–28, 2019.

[Ma *et al.*, 2020a] Meiyi Ma, Ezio Bartocci, Eli Lifland, John Stankovic, and Lu Feng. Sastl: Spatial aggregation signal temporal logic for runtime monitoring in smart cities. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 51–62, 2020.

[Ma *et al.*, 2020b] Meiyi Ma, Ji Gao, Lu Feng, and John Stankovic. Stlnet: Signal temporal logic enforced multivariate recurrent neural networks. *Advances in Neural Information Processing Systems*, 33:14604–14614, 2020.

[Ma *et al.*, 2021] Meiyi Ma, John Stankovic, Ezio Bartocci, and Lu Feng. Predictive monitoring with logic-calibrated uncertainty for cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 20(5s), September 2021.

[Maler and Nickovic, 2004] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *International symposium on formal techniques in real-time and fault-tolerant systems*, pages 152–166. Springer, 2004.

[McCoy *et al.*, 2024] R Thomas McCoy, Shunyu Yao, Dan Friedman, Mathew D Hardy, and Thomas L Griffiths. When a language model is optimized for reasoning, does it still show embers of autoregression? an analysis of openai o1. *arXiv preprint arXiv:2410.01792*, 2024.

[Meta, 2024] Meta. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, 2024.

[Miao *et al.*, 2023] Ning Miao, Yee Whye Teh, and Tom Rainforth. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. *arXiv preprint arXiv:2308.00436*, 2023.

[NY, 2025] NY. FDNY Issue Brief, 2025. Accessed: 2025-01-14.

[OpenAI, 2024a] OpenAI. Gpt-4o system card, 2024.

[OpenAI, 2024b] OpenAI. Openai o1 system card, 2024.

[Rouzegar and Makrehchi, 2024] Hamidreza Rouzegar and Masoud Makrehchi. Enhancing text classification through llm-driven active learning and human annotation. In *The 18th Linguistic Annotation Workshop (LAW-XVIII) Co-located with EACL 2024*, page 98, 2024.

[Shi *et al.*, 2023] Weijia Shi, Xiaodong Liu, Jing Shao, Pengcheng Liu, Jiawei Han, and Jianfeng Gao. Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*, 2023.

[Shuster *et al.*, 2022] Kurt Shuster, Samuel Humeau, Jing Xu, et al. Language models that seek for knowledge: Modular search and generation for dialogue and prompting. *arXiv preprint*, 2022.

[Toews *et al.*, 2021] Andrea J Toews, Donna E Martin, and Wanda M Chernomas. Clinical debriefing: a concept analysis. *Journal of clinical nursing*, 30(11-12):1491–1501, 2021.

[Turpin *et al.*, 2024] Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36, 2024.

[Wang *et al.*, 2024] Xiaohua Wang, Zhenghua Wang, Xuan Gao, Feiran Zhang, Yixin Wu, Zhibo Xu, Tianyuan Shi, Zhengyuan Wang, Shizheng Li, Qi Qian, et al. Searching for best practices in retrieval-augmented generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17716–17736, 2024.

[Wei *et al.*, 2022] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[Weng *et al.*, 2024] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Kang Liu, and Jun Zhao. Mastering symbolic operations: Augmenting language models with compiled neural networks. In *The Twelfth International Conference on Learning Representations*, 2024.

[Wu *et al.*, 2024] Xiaoqian Wu, Yong-Lu Li, Jianhua Sun, and Cewu Lu. Symbol-llm: leverage language models for symbolic system in visual human activity reasoning. *Advances in Neural Information Processing Systems*, 36, 2024.

[Zelikman *et al.*, 2022] Eric Zelikman, Yuhuai Wu, Timothy Novikoff, and Noah Goodman Li. Star: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.