DESDE 2023
https://rictrevista.org

RICT Revista de Investigación Científica, Tecnológica e Innovación

Publicación Semestral RICT Vol. 4 No. 7 (2026) P. 23 – 28

ISSN: 2992-7315

Análisis para la implementación de un sistema de soporte técnico automatizado basado en LLM y Docker

Analysis for the implementation of an automated technical support system based on LLM and Docker

Amaury-Castillo-Cruz [a] ,Emmanuel-Tonatihu-Juarez-Velázquez [a]

[m]*Maestria en Sistemas Computacionales, Tecnológico de Estudios Superiores de Ecatepec, 55210, Ecatepec de Morelos, Estado de México, México.*

Resumen

Se presenta el diseño e implementación de un sistema de soporte técnico automatizado para la atención de primer nivel, basado en un modelo de lenguaje de gran escala (LLM), una interfaz web en Python, una API de procesamiento y una arquitectura contenerizada con Docker. La propuesta automatiza la recepción, clasificación y respuesta inicial a tickets técnicos, y escala a soporte humano los casos complejos o de baja confianza. El estudio se estructuró como una investigación experimental-descriptiva orientada a evaluar precisión de clasificación, persistencia de datos y tiempos de respuesta bajo carga concurrente. En los escenarios controlados reportados, el sistema alcanzó una exactitud global de clasificación del 88% y un tiempo promedio de respuesta de 3.4 s con 20 solicitudes simultáneas, mientras que la reducción de memoria del contenedor del LLM a 2 GB elevó el tiempo promedio a 5.8 s. Los resultados sugieren que la arquitectura es viable para automatizar incidencias frecuentes y redistribuir la carga operativa de la mesa de ayuda. Se discuten además limitaciones metodológicas, de evaluación lingüística y de escalabilidad, así como líneas de trabajo futuro para robustecer el sistema.

Palabras clave: soporte técnico automatizado, LLM, Docker, clasificación de tickets, inteligencia artificial, arquitectura contenerizada

Abstract

This paper presents the design and implementation of an automated first-level technical support system based on a large language model (LLM), a Python web interface, an API layer, and a Docker-based containerized architecture. The proposal automates ticket intake, classification, and initial response generation, while escalating complex or low-confidence cases to human support. The study followed an experimental-descriptive design focused on classification accuracy, data persistence, and response time under concurrent workload. In the reported controlled scenarios, the system achieved an overall classification accuracy of 88% and an average response time of 3.4 s under 20 simultaneous requests; when the LLM container memory was reduced to 2 GB, the average response time increased to 5.8 s. The findings suggest that the proposed architecture is viable for automating recurrent incidents and redistributing help-desk workload. Methodological limitations, language-evaluation gaps, and future research directions are also discussed.

Keywords: automated technical support, LLM, Docker, ticket classification, artificial intelligence

Introducción

El crecimiento sostenido de los servicios digitales ha incrementado el volumen de solicitudes dirigidas a las mesas de ayuda, lo que ha tensionado los esquemas tradicionales de soporte basados exclusivamente en atención humana. En este contexto, los tiempos de respuesta, la saturación operativa y los costos de primer nivel se han convertido en factores críticos para la continuidad de los servicios y la experiencia del usuario.

Los modelos de lenguaje de gran escala (LLM) han mostrado capacidad para interpretar lenguaje natural, sintetizar información, clasificar textos y generar respuestas contextualizadas. Integrados dentro de una arquitectura controlada, estos modelos pueden funcionar como un filtro inteligente para resolver incidencias frecuentes, orientar al

*Autor para la correspondencia: 202510971@tese.edu.mx
Correo electrónico: 202510971@tese.edu.mx (Amaury-Castillo-Cruz),

Historial del manuscrito: recibido el 2/3/2026, última versión-revisada recibida el 21/4/2026, aceptado el 29/4/2026, publicado el 30/4/2026. **DOI:** https://doi.org/10.5281/zenodo.19889233

usuario y escalar oportunamente los casos de mayor complejidad.

No obstante, la incorporación de un chatbot por sí sola no garantiza una mejora operativa. La automatización del primer nivel de soporte requiere una arquitectura reproducible, auditable y escalable, capaz de separar servicios, registrar evidencia y mantener persistencia incluso ante reinicios o actualizaciones. Por ello, el presente trabajo propone una arquitectura contenerizada basada en Docker, una capa de procesamiento en Python y una base de datos PostgreSQL para soportar el ciclo completo del ticket.

El objetivo del estudio fue diseñar e implementar un sistema de soporte técnico automatizado capaz de: i) recibir solicitudes en lenguaje natural; ii) clasificar tickets por nivel de soporte; iii) generar respuestas iniciales contextualizadas; iv) persistir trazas del proceso; y v) escalar automáticamente los casos complejos. La contribución principal radica en articular la capa de inteligencia lingüística con una arquitectura de despliegue modular orientada a operación real.

Metodología

Diseño general de la solución

La solución se diseñó como una arquitectura de cuatro componentes principales: interfaz web, backend en Python, servicio de inferencia del LLM y base de datos PostgreSQL. Cada componente fue desplegado como un servicio independiente para facilitar aislamiento de dependencias, mantenimiento y crecimiento gradual del sistema. La Figura 1 presenta la arquitectura lógica propuesta.

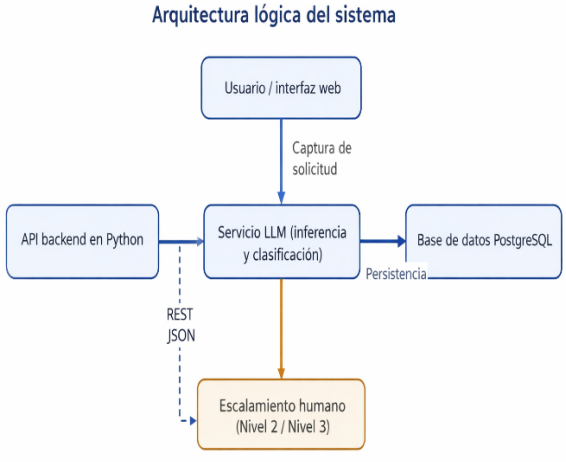

Figura 1. Arquitectura lógica del sistema de soporte técnico automatizado.

Modelo de lenguaje y estrategia de procesamiento

Se utilizó un LLM preentrenado basado en arquitectura Transformer, consumido como servicio independiente y operado mediante ingeniería de prompts. En la fase reportada no se describió un proceso de ajuste fino adicional; la adaptación al dominio se realizó mediante restricciones de salida, delimitación temática y reglas para forzar el escalamiento cuando la confianza fue baja o la consulta excedió el alcance del Nivel 1.

El flujo de procesamiento inició con la captura del incidente en lenguaje natural. Posteriormente, el backend realizó limpieza básica del texto, estructuración del prompt y envío al servicio del modelo. El LLM generó una respuesta inicial y una categoría de soporte, que posteriormente fue persistida junto con la evidencia del ticket. El proceso general se resume en la Figura 4.

Despliegue contenerizado

La implementación se realizó con Docker y Docker Compose. El backend, el servicio del LLM y PostgreSQL se ejecutaron en contenedores separados, comunicados por una red interna tipo bridge. Esta separación permitió encapsular dependencias, facilitar reinicios controlados y evitar conflictos de configuración entre servicios. La base de datos se configuró con volúmenes persistentes para conservar los registros aun cuando los contenedores fueran recreados.

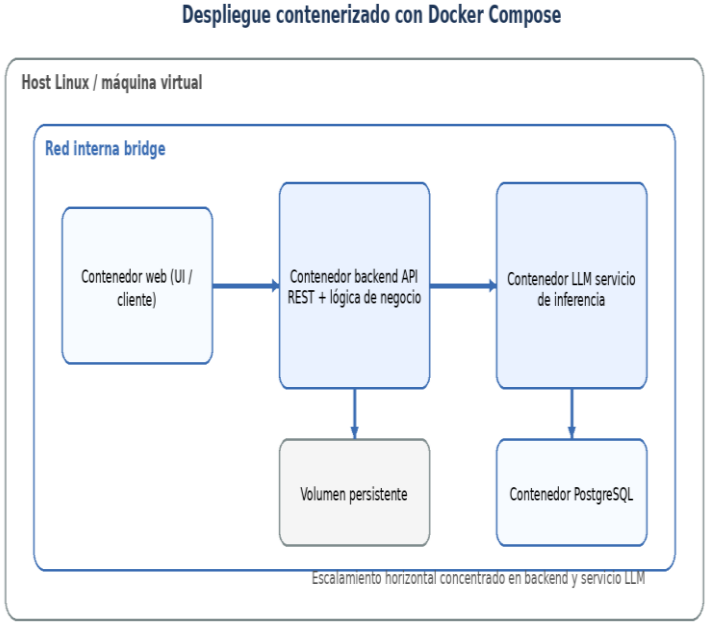

Figura 2. Despliegue contenerizado mediante Docker Compose y red interna de servicios.

Modelo de datos y trazabilidad

Con el fin de garantizar trazabilidad, auditoría y análisis posterior, se diseñó una estructura relacional en PostgreSQL compuesta por entidades para usuarios, tickets, respuestas generadas por el modelo, niveles de soporte e historial de estados. De esta manera, cada ticket conserva su descripción original, la respuesta automatizada, el nivel asignado y la evolución del flujo operativo.

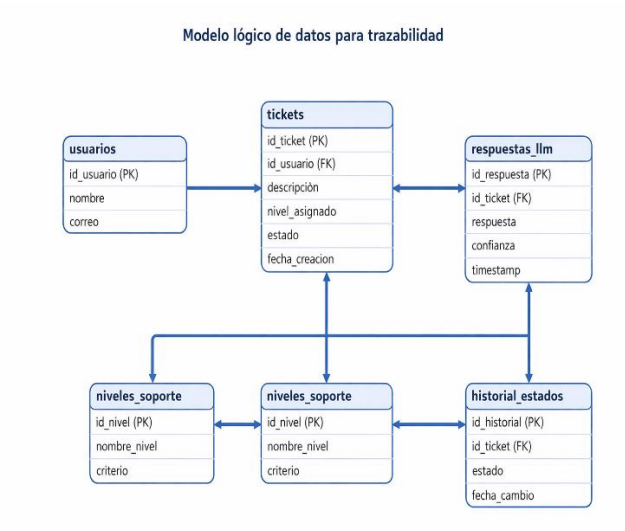

Figura 3. Modelo lógico de datos para el registro y seguimiento del ticket.

Diseño experimental

El estudio se planteó bajo un enfoque experimental-descriptivo. Se realizaron pruebas controladas para analizar el comportamiento del sistema en tres dimensiones: clasificación automática de tickets, tiempo de respuesta y persistencia de datos. Los tickets utilizados en la evaluación incluyeron incidencias frecuentes del Nivel 1 (por ejemplo, restablecimiento de contraseña, configuración básica y dudas operativas) e incidencias complejas asociadas a fallos de red o infraestructura, que debían ser escaladas al Nivel 2 o Nivel 3.

Elemento experimental	Descripción
Tipo de estudio	Experimental-descriptivo.
Unidad de análisis	Tickets de soporte técnico procesados por el sistema.
Conjunto de prueba	Pruebas realizadas en conjunto con el servidor
Etiquetado de referencia	Tickets del sistema aplicados
Escenarios evaluados	Clasificación por nivel, persistencia de datos y carga concurrente.
Cargas ejecutadas	1 usuario, 20 usuarios concurrentes y escenario degradado con 2 GB de memoria para el contenedor del LLM.
Cargas sugeridas para reenvío	50 y 100 usuarios concurrentes para validar escalabilidad real.
Entorno de hardware	Windows 11 , 32 GB RAM , RTX 5060 ,Docker Desktop

Tabla 1. Configuración metodológica y elementos mínimos de replicabilidad.

Métricas de evaluación

Para la dimensión de clasificación, la métrica central reportada fue la exactitud global (accuracy). No obstante, para una validación científica más robusta se requiere complementar el análisis con precisión, recall, F1-score por clase y matriz de confusión, dado que la clasificación por niveles de soporte constituye una tarea multiclase donde el costo operativo del error no es uniforme.

Para la dimensión lingüística, la calidad de la respuesta automatizada debería evaluarse con dos enfoques complementarios: i) métricas automáticas de similitud textual, como BLEU y ROUGE, cuando exista una respuesta de referencia; y ii) evaluación humana basada en pertinencia, claridad, utilidad técnica y adecuación al dominio. Este segundo componente es especialmente importante porque una respuesta con alta similitud léxica no necesariamente implica utilidad operativa.

Resultados

Los resultados obtenidos muestran que la integración entre el backend, el servicio del LLM y PostgreSQL fue funcional durante las pruebas controladas. El sistema logró generar respuestas iniciales para incidencias frecuentes, registrar la información asociada al ticket y escalar los casos complejos cuando la consulta rebasó el dominio previsto para el primer nivel.

Variable	Resultado reportado	Observación técnica
Exactitud global de clasificación	88%	Valor preliminar. Debe complementarse con precisión, recall, F1-score y matriz de confusión.

Tiempo promedio con 20 solicitudes concurrentes	3.4 s	Desempeño aceptable en carga ligera concurrente.
Tiempo promedio con el contenedor del LLM limitado a 2 GB	5.8 s	Evidencia sensibilidad del servicio del LLM a la memoria asignada.
Persistencia de datos tras reinicio de contenedores	Conservada	Los volúmenes de Docker mantuvieron la información sin pérdida.
Clasificación de incidencias frecuentes	Correcta en la mayoría de los escenarios controlados	Incluyó casos típicos de Nivel 1.
Escalamiento de casos complejos	Activado hacia Nivel 2 / Nivel 3	Observado en incidencias de red e infraestructura.
Evaluación lingüística de las respuestas	Respuestas Interactivas de aprendizaje continuo	Agregar BLEU/ROUGE si existen respuestas de referencia y evaluación humana con escala definida.

Tabla 2. Resultados experimentales reportados en la presente versión del manuscrito.

Desde la perspectiva operativa, el hallazgo más relevante fue la capacidad del sistema para absorber incidencias frecuentes del Nivel 1 sin intervención humana inmediata. Ello sugiere un potencial de redistribución de carga hacia tareas especializadas. Sin embargo, el valor de exactitud reportado debe interpretarse como preliminar, pues no se acompañó de una matriz de confusión ni de métricas por clase.

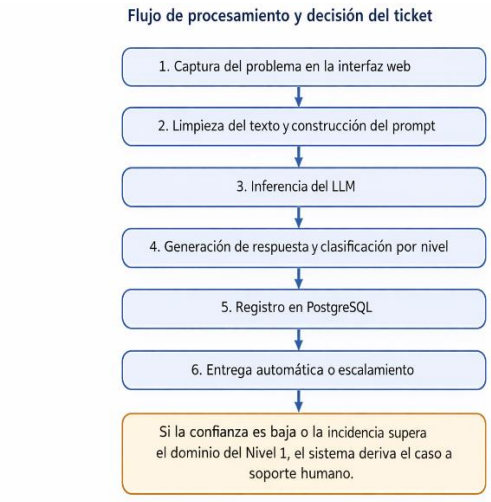

Figura 4. Flujo de procesamiento del ticket desde la captura hasta la respuesta o el escalamiento.

Comparación con enfoques alternativos

Con el fin de contextualizar la propuesta, se realizó una comparación cualitativa con dos familias de soluciones existentes: los chatbots tradicionales basados en reglas y las plataformas comerciales de atención como Zendesk y Freshdesk. La comparación no pretende demostrar superioridad absoluta, sino ubicar el aporte del sistema dentro del ecosistema actual de soluciones.

Criterio	Sistema propuesto	Chatbot basado en reglas	Plataformas comerciales
Comprensión del lenguaje natural	Alta; depende del LLM y del prompt.	Limitada a reglas fijas.	Alta; IA y automatización maduras.
Control de arquitectura y despliegue	Alto; modular y adaptable.	Alto en flujos simples.	Menor; depende del proveedor.
Escalamiento automático	Sí, por complejidad o baja confianza.	Por reglas fijas.	Sí, con routing y flujos integrados.
Costo inicial de implementación	Moderado; requiere desarrollo e infraestructura.	Bajo a moderado.	Licenciamiento por usuario e integración.
Madurez funcional	Prototipo funcional validado de forma preliminar.	Útil para FAQ y flujos acotados.	Alta madurez e integración empresarial.
Adecuación para investigación	Alta; permite experimentar con arquitectura y métricas.	Media; menor riqueza semántica.	Media; alta capacidad operativa, menor control experimental.

Tabla 3. Comparación cualitativa entre el sistema propuesto y soluciones alternativas.

Discusión

Los resultados respaldan la viabilidad técnica de integrar un LLM dentro de una arquitectura contenerizada para automatizar parte del primer nivel de atención. La exactitud global reportada y los tiempos observados bajo carga ligera sugieren que la propuesta es funcional para incidencias recurrentes, siempre que el dominio del problema esté acotado y que existan criterios claros de escalamiento.

A diferencia de un chatbot estrictamente basado en reglas, el sistema propuesto ofrece mayor flexibilidad semántica para interpretar consultas expresadas de forma natural y heterogénea. No obstante, esa ventaja introduce un comportamiento probabilístico que exige mecanismos explícitos de contención: instrucciones de dominio, límites de respuesta, reglas de derivación y supervisión de calidad.

En comparación con plataformas comerciales consolidadas, la principal fortaleza de la propuesta reside en

el control arquitectónico y en la posibilidad de adaptar el despliegue a necesidades específicas de investigación o de operación local. Su principal debilidad, por el contrario, se encuentra en la menor madurez funcional: no dispone aún de ecosistemas de integración, analítica avanzada, gobierno operacional ni validación extensiva comparable con soluciones empresariales establecidas.

Desde el punto de vista metodológico, la mayor debilidad del manuscrito original no fue la ausencia de resultados, sino la insuficiente formalización del experimento. Reportar únicamente una exactitud global y observaciones descriptivas no permite estimar con precisión la calidad del clasificador ni la utilidad real de las respuestas generadas. Por ello, la siguiente etapa del estudio debe incorporar una evaluación por clase, carga extendida de 50 y 100 usuarios concurrentes, análisis de dispersión temporal y juicio experto sobre la calidad de las respuestas.

Limitaciones y trabajo futuro

El estudio presenta varias limitaciones. En primer lugar, la versión inicial del manuscrito no especificó con precisión el modelo utilizado, la versión, la configuración de inferencia ni el entorno de hardware, lo que dificulta la replicabilidad. En segundo lugar, la evaluación lingüística de las respuestas no fue formalizada mediante BLEU, ROUGE o evaluación humana estructurada. En tercer lugar, la validación de escalabilidad se limitó a 20 usuarios concurrentes, por lo que no es posible afirmar desempeño estable para cargas más altas.

Adicionalmente, la exactitud global de 88% no fue acompañada por precisión, recall, F1-score ni matriz de confusión, lo que impide identificar en qué niveles de soporte se concentra el error. Tampoco se reportó un análisis estadístico de dispersión de tiempos ni un estudio de costo-beneficio frente a la operación manual.

Como líneas de trabajo futuro se proponen: i) ampliar las pruebas a 50, 100 o más usuarios concurrentes; ii) incorporar evaluación humana de la utilidad técnica de las respuestas; iii) explorar mecanismos de grounding mediante bases de conocimiento o recuperación aumentada; iv) comparar el desempeño con un chatbot basado en reglas en un mismo conjunto de tickets; y v) estudiar estrategias de ajuste fino o especialización del modelo para dominios técnicos concretos.

Conclusiones

Se diseñó e implementó un sistema funcional para la automatización del primer nivel de soporte técnico, articulando una interfaz web en Python, un servicio LLM, una API de procesamiento, PostgreSQL y contenedores Docker. La propuesta demostró capacidad para clasificar tickets, generar respuestas iniciales y escalar casos complejos dentro de una arquitectura modular y persistente.

Los resultados preliminares muestran que el sistema puede reducir tiempos de atención y redistribuir la carga operativa hacia niveles especializados. Sin embargo, la evidencia reportada debe entenderse como una validación inicial y no como una demostración concluyente de superioridad frente a otras soluciones. La consolidación científica del trabajo exige completar los datos metodológicos pendientes y ampliar la evaluación experimental.

Aun con estas reservas, el estudio aporta una base arquitectónica y metodológica útil para futuras investigaciones sobre automatización del soporte técnico con inteligencia artificial, especialmente en escenarios donde se requiere control del despliegue, trazabilidad del proceso y adaptación del sistema a políticas operativas locales.

Agradecimientos

El autor agradece al Tecnológico de Estudios Superiores de Ecatepec por el respaldo académico brindado durante el desarrollo del trabajo, así como a los asesores y revisores que, mediante sus observaciones, contribuyeron al fortalecimiento metodológico y técnico del manuscrito.

Referencias

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.

Davenport, T. H., & Ronanki, R. (2018). Artificial intelligence for the real world. Harvard Business Review, 96(1), 108–116.

Freshworks. (2026). Freshdesk: AI-powered platform for modern customer service. Recuperado el 19 de marzo de 2026 de https://www.freshworks.com/freshdesk/

Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media.

Huang, M. H., & Rust, R. T. (2021). Artificial intelligence in service. Journal of Service Research, 24(1), 3–16. https://doi.org/10.1177/1094670520902266

ITIL. (2019). ITIL Foundation: ITIL 4 Edition. AXELOS.

Lin, C.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. In Proceedings of the Workshop on Text Summarization Branches Out (pp. 74–81). Association for Computational Linguistics.

Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014(239), 2.

Newman, S. (2015). Building microservices. O'Reilly Media.

Norman, D. (2013). The design of everyday things. Basic Books.

OpenAI. (2023). GPT-4 technical report. arXiv:2303.08774.

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). BLEU: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (pp. 311–318). Association for Computational Linguistics.

Richards, M., & Ford, N. (2020). Fundamentals of software architecture. O'Reilly Media.

Van Rossum, G., & Drake, F. L. (2009). Python 3 reference manual. CreateSpace.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30.

Zendesk. (2026). AI for customer service. Recuperado el 19 de marzo de 2026 de https://www.zendesk.com/service/ai/.

Agarwal, S., Sindhgatta, R., & Sengupta, B. (2012). SmartDispatch: Enabling efficient ticket dispatch in an IT service environment. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM. doi:10.1145/2339530.2339744.

Al-Hawari, F., & Barham, H. (2021). A machine learning based help desk system for IT service management. Journal of King Saud University - Computer and Information Sciences, 33(6), 702–718. doi:10.1016/j.jksuci.2019.04.001.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of NAACL-HLT 2019 (pp. 4171–4186). Association for Computational Linguistics.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2024). Retrieval-augmented generation for large language models: A survey. arXiv. doi:10.48550/arXiv.2312.10997.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. arXiv. doi:10.48550/arXiv.2005.11401.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-rank adaptation of large language models. arXiv. doi:10.48550/arXiv.2106.09685.

Marcuzzo, M., Zangari, A., Schiavinato, M., Giudice, L., Gasparetto, A., & Albarelli, A. (2022). A multi-level approach for hierarchical ticket classification. In Proceedings of the Eighth Workshop on Noisy User-generated Text (W-NUT 2022) (pp. 201–214). Association for Computational Linguistics.

Olujimi, P. A., & Ade-Ibijola, A. (2023). NLP techniques for automating responses to customer queries: A systematic review. Discover Artificial Intelligence, 3, 20. doi:10.1007/s44163-023-00065-5.