Modelos de Lenguaje Pequeños para la Interpretación de Comandos de Domótica en Español: Evaluación Automática de Doce Modelos Sub-2B

Carlos Binker, Lautaro Lasorsa, Hugo Tantignone, Guillermo Buranits, Eliseo Zurdo, and Maximiliano Frattini

Universidad Nacional de La Matanza, Florencio Varela 1903 (B1754JEC), San Justo, Buenos Aires, Argentina
{cbinker, laulasorsa, htantignone, gburanits, eazurdo, mfrattini}@unlam.edu.ar

Resumen Los asistentes de domótica basados en modelos de lenguaje suelen depender de servicios en la nube, lo que introduce alta latencia, costos recurrentes y dependencia de conectividad. Los Small Language Models (SLMs) ejecutados localmente constituyen una alternativa viable para hubs domóticos de bajo costo, pero su capacidad para interpretar comandos en español rioplatense no ha sido evaluada: el voseo, los diminutivos y las expresiones relativas de la variedad ("bájale un toque") no aparecen en los corpus de domótica en inglés usados para entrenar y evaluar estos modelos, y su tratamiento no puede suponerse transferido. Este trabajo evalúa doce SLMs abiertos de menos de 2 mil millones de parámetros sobre un dataset de 32 comandos en español rioplatense, con anotación de referencia de cinco campos (intención, dispositivo, ubicación, valor y unidad). La verificación es automática en dos etapas: comparación textual determinista campo a campo y un modelo de lenguaje que actúa de juez, clasificando cada respuesta incorrecta contra una taxonomía cerrada. Se reportan exactitud estricta y exactitud laxa (semántica): su brecha cuantifica cuánto penaliza la coincidencia textual exacta a respuestas que un evaluador humano consideraría correctas.

Keywords: modelos de lenguaje pequeños, domótica, evaluación automática, LLM como juez, procesamiento del lenguaje natural, español rioplatense.

1. Introducción

Los asistentes de voz y los sistemas domóticos interpretan comandos en lenguaje natural ("prendé la luz del living") casi siempre con modelos de gran escala en la nube, con latencia de red, costo recurrente por uso de API y dependencia de conectividad. Los Small Language Models (SLMs) pueden ejecutarse en el propio hub domótico y alcanzan, en tareas acotadas, un desempeño competitivo

frente a modelos mucho más grandes [1], pero la literatura los evalúa mayormente en inglés. Además, clasificar manualmente sus respuestas incorrectas no escala a doce modelos ni es reproducible por terceros.

Este trabajo aporta: (1) la evaluación de doce SLMs abiertos sub-2B, sin restricción de licencia ni de token, en dos franjas de seis modelos bajo un protocolo común y reproducible, con aislamiento por contenedor y versiones documentadas por grupo; (2) un procedimiento de verificación automática en dos etapas — coincidencia textual determinista más un juez de lenguaje con taxonomía cerrada — que elimina todo etiquetado manual; (3) el reporte conjunto de exactitud estricta y laxa, cuya brecha cuantifica el costo de exigir coincidencia textual exacta; y (4) la inclusión deliberada de `Qwen2.5-1.5B-Instruct` como representante de la generación anterior de SLMs, medido bajo el mismo protocolo.

2. Trabajos relacionados

Convertir un comando en una estructura de intención y argumentos (*slots*) es una tarea clásica de Natural Language Understanding (NLU): Weld et al. [2] revisan los modelos conjuntos de detección de intención y *slot filling* previos a los LLMs, entrenados específicamente para el dominio; el enfoque actual genera directamente un JSON con un modelo general. Desot et al. [3] mostraron, sobre un corpus de comandos domóticos en francés, que los recursos de NLU en inglés no se transfieren bien a otras lenguas. El antecedente más cercano evalúa LLMs en el dispositivo para asistentes domóticos [4]: variantes cuantizadas de 16, 8 y 4 bits, con 80 %–86 % de exactitud y latencias de 5 a 6 segundos. Este trabajo se diferencia por el tamaño (sub-2B, sin cuantización), el idioma y la interpretación estructurada en lugar de la generación conversacional.

Gupta et al. [1] documentan que modelos de pocos miles de millones de parámetros igualan a modelos mucho mayores en tareas específicas; los reportes técnicos de Qwen2.5 [5], SmolLM2 [6] y OLMo 2 [7] describen tres de las familias evaluadas. La decodificación restringida por gramática garantiza validez sintáctica [8,9]; no se la empleó aquí, para medir el seguimiento de instrucciones en un despliegue simple y porque esa garantía cubre el formato sin impedir un valor incorrecto dentro de un JSON válido.

Zheng et al. [10] muestran que un LLM capaz aproxima el juicio humano con acuerdo alto, y proponen usarlo como juez donde la coincidencia exacta subestima el desempeño real, como en la brecha estricta/laxa de este trabajo (Sección 3.4). Su limitación conocida, el sesgo de auto-favorecimiento, es directamente aplicable aquí porque el juez es uno de los modelos evaluados (Sección 6).

3. Metodología

3.1. Modelos evaluados

Se evaluaron doce modelos abiertos, públicos en HuggingFace y sin restricción de licencia ni de token, en dos franjas de seis: sub-1B y de 1 a 2B (Tabla

Tabla 1. Modelos evaluados y versión de `transformers` resuelta; la línea separa las franjas sub-1B y 1–2B. Versión fijada por: `tokenizer` no soportado en 4.57.6 (`LFM2.5-230M`); idem (`LFM2.5-350M`); regresión de caché en 5.14.1 (`granite-4.0-350m`); arquitectura no reconocida en 4.57.6 (`Qwen3.5-0.8B`).

Modelo	Parámetros	Familia	`transformers`
LFM2.5-230M	0,23B	LiquidAI	5.14.1
granite-4.0-h-350m	0,34B	IBM Granite	4.57.6
LFM2.5-350M	0,35B	LiquidAI	5.14.1
granite-4.0-350m	0,35B	IBM Granite	4.57.6
SmolLM2-360M-Instruct	0,36B	HuggingFaceTB	4.57.6
Qwen3.5-0.8B	0,80B	Qwen	5.14.1
OLMo-2-0425-1B-Instruct	1,00B	AllenAI	4.57.6
LFM2.5-1.2B-Instruct	1,20B	LiquidAI	4.57.6
granite-4.0-h-1b	1,50B	IBM Granite	4.57.6
Qwen2.5-1.5B-Instruct	1,54B	Qwen	4.57.6
granite-4.0-1b	1,60B	IBM Granite	4.57.6
SmolLM2-1.7B-Instruct	1,71B	HuggingFaceTB	4.57.6

1). `Qwen2.5-1.5B-Instruct` (1,54 B) se incorporó deliberadamente como representante de la generación anterior de SLMs bajo el mismo protocolo (Sección 4.1).

3.2. Dataset de comandos

El dataset consta de 32 comandos de domótica en español rioplatense (voseo), con una anotación de referencia de cinco campos: `intent`, `dispositivo`, `ubicacion`, `valor` y `unidad`. Cada comando pertenece además a una de seis categorías lingüísticas cerradas — encendido/apagado simple, ajuste con valor numérico explícito, ajuste relativo o cualitativo, múltiples dispositivos, consulta de estado y negación o contexto conversacional —, que asigna el juez automático (Sección 3.5) y no un anotador humano.

3.3. Protocolo de prompting y esquema de salida

El prompt de sistema es único y fijo para los doce modelos, aplicado mediante la plantilla de conversación del propio *tokenizer* de cada uno. Se transcribe íntegro junto con el prompt de usuario; ambos provienen de constantes del código, no de una transcripción manual.

Prompt de sistema (texto completo, transcripción literal del código: la ausencia de tildes es intencional).

```
Sos un módulo de interpretación de comandos de voz para un sistema
domótico en español rioplatense. Convertí el comando del usuario en UN
objeto JSON con estos campos: "intent"
(encender/apagar/ajustar/consultar), "dispositivo" (luz,
```

```
aire_acondicionado, calefaccion, persiana, cortina, tv, parlante,
cerradura, ventilador, todos), "ubicacion" (living, cocina, dormitorio,
bano, garage, patio, entrada, todas), "valor" (número si el comando lo
da, si no null), "unidad" ("%" o "°C" si corresponde, si no null).
Reglas: respondé SOLO el JSON, sin texto adicional; si el comando ajusta
sin dar un número, "valor" y "unidad" deben ser null.
Ejemplo: Comando: "Poné el aire de la cocina en 20 grados."
{"intent": "ajustar", "dispositivo": "aire_acondicionado", "ubicacion":
"cocina", "valor": 20, "unidad": "°C"}
IMPORTANTE: los campos "intent", "dispositivo", "ubicacion" y "unidad"
son campos CATEGORICOS CERRADOS. Los valores listados entre parentesis
mas arriba son, textualmente, los unicos outputs aceptados para esos
campos: copialos exactamente como aparecen, sin tildes, sin mayusculas,
sin plural y sin reformular. Usar un sinonimo, una traduccion o
cualquier variante que no este literalmente en esa lista (por ejemplo
"televisor" o "tele" en lugar de "tv") es una violacion del formato y la
respuesta se considera incorrecta. Si ningun valor de la lista aplica,
usa null.
Ejemplo 2: Comando: "Bajale un poco a la luz del living."
{"intent": "ajustar", "dispositivo": "luz", "ubicacion": "living",
"valor": null, "unidad": null}
```

Prompt de usuario (uno por comando, tras el de sistema).

```
Comando: "<comando del usuario>"
```

El prompt solicita un único objeto JSON con los cinco campos, incluye dos ejemplos *few-shot* y agrega una cláusula taxativa sobre los campos categóricos cerrados: los valores listados entre paréntesis son, textualmente, las únicas salidas aceptadas, y un sinónimo válido en español —"tele" por "tv"— constituye una violación de formato. Los resultados de la Sección 4 se calculan bajo este prompt, por lo que la comparación entre arquitecturas nunca mezcla prompts.

3.4. Métricas

La **exactitud estricta** exige que los cinco campos coincidan textualmente con la referencia bajo ese protocolo taxativo; la **exactitud laxa (semántica)** incorpora además los casos que el juez etiquetó como `sin_error_semantico` o `uso_de_sinonimos`. La estricta es la métrica primaria; la laxa es una cota superior que separa errores de forma de errores de contenido, y la brecha entre ambas es un resultado del estudio, no una corrección que deba promediarse. Se reportan además la tasa de JSON válido, la exactitud por campo y por categoría, y la latencia promedio.

3.5. Verificación automática en dos etapas

En la **etapa 1** cada respuesta se compara campo a campo, de forma determinista, contra la referencia, sin ningún modelo adicional, y de ahí sale la exactitud

estricta. El juez de la **etapa 2** se determina por una regla fijada de antemano: el modelo con mayor exactitud estricta en la etapa 1, con empates resueltos por mayor cantidad de parámetros y luego por el orden del listado. Ambas etapas usan decodificación *greedy* (`do_sample=False`, temperatura 0); de eso — no de fijar el identificador del juez de antemano — depende la reproducibilidad.

El juez clasifica cada respuesta con `match_exact = False` en un subconjunto no vacío de siete etiquetas cerradas — confusión de intención, de dispositivo y de ubicación; alucinación de valor o unidad; valor numérico incorrecto; más `sin_error_semantico` y `uso_de_sinonimos`, que definen la exactitud laxa — y asigna a cada comando exactamente una categoría lingüística. Una salida que no valide contra ese vocabulario se reintenta una vez y, si vuelve a fallar, recibe una etiqueta de respaldo determinista; su frecuencia se reporta. Al ser uno de los modelos evaluados, el juez adjudica también sobre sus propias respuestas (Sección 6).

3.6. Entorno de ejecución y reproducibilidad

El host es un equipo con procesador Intel Core i5-11400H (6 núcleos físicos / 12 hilos), sin GPU: los doce modelos y el juez se ejecutan en CPU, cada uno en su propia imagen de contenedor y de forma secuencial.

El conjunto evaluado necesita **dos grupos de versión** de `transformers`, mutuamente excluyentes (Tabla 1): 4.57.6 para nueve modelos y 5.14.1 para los otros tres. `granite-4.0-350m` falla bajo 5.14.1, y `LFM2.5-230M`, `LFM2.5-350M` y `Qwen3.5-0.8B` fallan bajo 4.57.6: el segundo grupo se resuelve precisamente a la versión que hace fallar al modelo que impone el primero, de modo que ninguna versión mayor única cubre el conjunto (Sección 6).

La configuración de Docker de esta máquina expone únicamente **dos procesadores lógicos** y un límite de **8 GB**, y las trece ejecuciones — doce del barrido más la del juez — fijan `--memory=8g --cpus=2 --cpuset-cpus=0-1`, el mismo par de identificadores lógicos en todas. Que ese par corresponda a un mismo núcleo físico por *hyperthreading* (SMT) o a dos núcleos distintos lo decide el hipervisor y no es observable desde el contenedor. La ejecución es secuencial porque la inferencia en CPU está limitada por el ancho de banda de memoria, que no se particiona por contenedor. Ninguno de estos factores afecta a la exactitud, calculada con decodificación determinista, pero sí a las latencias: comparables entre los doce modelos, aunque sus valores absolutos no sean los mejores alcanzables en CPU[1] (Sección 6).

4. Experimentación y resultados

4.1. Resultados globales

El barrido se completó sin fallas irrecuperables: la Tabla 2 resume las 384 respuestas completas (12 modelos × 32 comandos). `granite-4.0-1b` alcanza

[1] Código y datos: `https://github.com/carlosbinker/cacic-2026` (rama `feat/reescritura-experimento-2026`).

Tabla 2. Resultados globales por modelo (JSON válido, exactitud estricta y laxa, latencia) y distribución de errores de la taxonomía. I: incorrectas; CI: confusión de intención; CD: confusión de dispositivo; CU: confusión de ubicación; VN: valor numérico incorrecto; SE: sin error semántico (equivalente por otro motivo); US: uso de sinónimos (difiere solo por un sinónimo).

Modelo	JSON	Estr.	Laxa	Lat.	I	CI	CD	CU	VN	SE	US
LFM2.5-230M	87,5	6,2	34,4	6,31	30	0	20	4	0	1	8
granite-4.0-h-350m	100,0	40,6	71,9	56,45	19	0	9	0	0	3	7
LFM2.5-350M	100,0	0,0	37,5	9,62	32	0	20	5	0	1	11
granite-4.0-350m	100,0	28,1	62,5	13,77	23	3	9	0	0	5	6
SmolLM2-360M-Instruct	25,0	9,4	93,8	20,47	29	0	2	0	0	1	26
Qwen3.5-0.8B	100,0	75,0	84,4	18,17	8	1	4	0	0	0	3
OLMo-2-0425-1B-Instruct	100,0	3,1	37,5	35,31	31	0	19	3	1	2	9
LFM2.5-1.2B-Instruct	100,0	56,2	84,4	31,88	14	2	3	0	0	2	7
granite-4.0-h-1b	100,0	71,9	96,9	123,15	9	0	1	0	0	7	1
Qwen2.5-1.5B-Instruct	100,0	65,6	93,8	38,13	11	0	1	1	0	2	7
granite-4.0-1b	100,0	90,6	100,0	42,78	3	0	0	0	0	1	2
SmolLM2-1.7B-Instruct	100,0	50,0	84,4	56,71	16	0	5	0	0	5	6

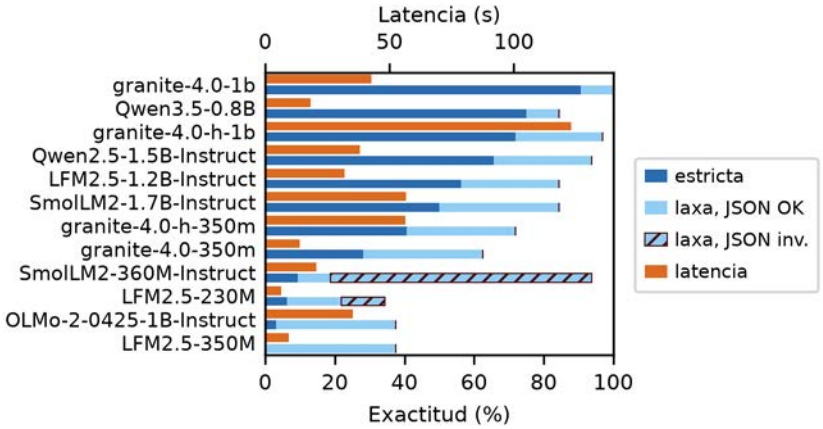


Figura 1. Exactitud y latencia por modelo, cada una contra su propio eje. La barra acumula estricta, laxa con JSON válido y, en trama rayada, laxa con JSON inválido, en orden de estricta decreciente.

90,6 % de exactitud estricta — la cifra más alta del conjunto — frente al **65,6 %** de `Qwen2.5-1.5B-Instruct`, la generación anterior: **+25,0 puntos porcentuales**. En el otro extremo, `LFM2.5-350M` no acierta ninguno de los 32 comandos (0,0 %) pese a emitir JSON válido en el 100 % de sus respuestas. La ventaja de la franja 1–2B no es universal: `Qwen3.5-0.8B` supera a cuatro de los seis modelos de la franja superior.

La brecha entre exactitud estricta y laxa es amplia y heterogénea (Figura 1): en varios modelos, la laxa más que duplica la estricta. Por campo del esquema,

`intent` y `unidad` concentran la mayor tasa de error, y `valor` y `ubicacion`, la menor.

4.2. Exactitud por categoría lingüística

Tabla 3. Exactitud estricta por categoría lingüística

Modelo	Encendido / apagado simple (15)	Ajuste con valor numérico (14)	Ajuste relativo / cualitativo (3)
LFM2.5-230M	6,7	7,1	0,0
granite-4.0-h-350m	6,7	78,6	33,3
LFM2.5-350M	0,0	0,0	0,0
granite-4.0-350m	20,0	35,7	33,3
SmolLM2-360M-Instruct	6,7	7,1	33,3
Qwen3.5-0.8B	60,0	92,9	66,7
OLMo-2-0425-1B-Instruct	0,0	7,1	0,0
LFM2.5-1.2B-Instruct	46,7	71,4	33,3
granite-4.0-h-1b	60,0	92,9	33,3
Qwen2.5-1.5B-Instruct	60,0	78,6	33,3
granite-4.0-1b	86,7	100,0	66,7
SmolLM2-1.7B-Instruct	33,3	71,4	33,3

De las seis categorías cerradas del dataset, el juez usó solo tres (Tabla 3): *encendido/apagado simple* (15 comandos), *ajuste con valor numérico* (14) y *ajuste relativo o cualitativo* (3). Las consultas de estado, las negaciones y los comandos sobre varios dispositivos se absorbieron en las dos primeras. Ese colapso no afecta la exactitud (se calcula por comando) pero limita esta tabla como desagregación lingüística fina.

4.3. Análisis automático de errores

El juez, por la regla de la Sección 3.5, es `granite-4.0-1b` (**90,6 %** de exactitud estricta en etapa 1). Clasificó las 225 respuestas con `match_exact = False` en las siete etiquetas cerradas sin un solo `juez_parse_ok = False`, ni siquiera en las 32 asignaciones de categoría. No hubo reintentos ni etiquetas de respaldo. De esas siete, *alucinación de valor/unidad* nunca se instanció — ningún modelo produjo valores ni unidades inexistentes, por eso no aparece como columna en la Tabla 2 — y *valor numérico incorrecto* solo una vez, en `OLMo-2-0425-1B-Instruct`. Los errores se concentran en confusión de dispositivo y sinónimos; la métrica laxa acredita **123** de las 225 incorrectas.

Una exactitud estricta cercana a cero admite dos lecturas, distinguibles por `json_valido`: alto indica un fallo semántico genuino (formato correcto, contenido no); bajo, un artefacto de parseo. `LFM2.5-350M` tiene 0,0 % de exactitud estricta con **100 %** de JSON válido y `OLMo-2-0425-1B-Instruct` 3,1 % con también **100 %**: ambos son fallos semánticos. `SmolLM2-360M-Instruct`, en cambio,

tiene 9,4 % con apenas **25 %**; solo nueve modelos llegan al 100 % y `LFM2.5-230M` al 87,5 %.

El campo `laxo_json_invalido_n` (respuestas laxamente correctas pese a JSON inválido) mide esa fragilidad de formato en la métrica laxa: **24** para `SmolLM2-360M-Instruct` sobre apenas **27** recuperadas — su exactitud laxa de 93,8 % es un artefacto, no evidencia de competencia semántica —, **4** para `LFM2.5-230M` y **0** para los diez modelos restantes.

5. Discusión

La brecha entre exactitud estricta y laxa mide cuánto penaliza la coincidencia textual exacta a respuestas que un juez semántico considera equivalentes; la taxonomía la descompone en dos causas: ausencia de error semántico y sustitución de un valor del vocabulario cerrado por un sinónimo.

Endurecer el prompt no elimina el problema. Pese a la cláusula taxativa del prompt (Sección 3.3), `uso_de_sinonimos` aparece en casi todos los modelos: copiar textualmente un token de una lista cerrada exige un seguimiento de instrucciones distinto de acertar el contenido, y varios modelos con buena exactitud por campo no logran la forma literal. El caso extremo es `SmolLM2-360M-Instruct`, cuyas respuestas incorrectas se explican casi por completo por esta única causa (Tabla 2).

Tamaño, arquitectura y generación. Que `Qwen3.5-0.8B` (menos de mil millones de parámetros) supere a cuatro de los seis modelos de la franja superior indica que el desempeño no depende solo del tamaño; los +25,0 puntos de `granite-4.0-1b` sobre la generación anterior cuantifican esa distancia en el rango sub-2B.

Penalización de latencia de la atención híbrida. `granite-4.0-h-350m` tarda **56,45** segundos por comando frente a los **13,77** de la variante densa `granite-4.0-350m` (×**4,10**), y `granite-4.0-h-1b` **123,15** frente a los **42,78** de `granite-4.0-1b` (×**2,88**). Que se reproduzca en ambos pares, misma familia y protocolo, indica un efecto arquitectónico, no un artefacto puntual. Las híbridas se proponen por su complejidad sub-cuadrática, que solo rinde en secuencias largas; aquí los prompts son cortos y la ejecución es en CPU, así que no se manifiesta, mientras que la sobrecarga constante de gestionar dos tipos de caché — atención estándar y estado lineal/recurrente — domina el tiempo total. A igual tamaño, la variante densa es preferible en CPU para prompts cortos.

Viabilidad práctica. Los 42,78 segundos por comando del modelo más exacto son incompatibles con una interacción de voz en tiempo real, aunque resultan aceptables para automatización diferida; entre los sub-1B, `Qwen3.5-0.8B` alcanza la mayor exactitud estricta (75,0 %) con 18,17 segundos.

6. Amenazas a la validez

Constructo. La exactitud laxa y la taxonomía separan qué parte de la exactitud estricta perdida es de vocabulario y qué parte es de contenido, pero

solo resuelven la amenaza parcialmente: la brecha es poco confiable donde el formato es débil (Sección 4.3).

Interna. *Sesgo de auto-favorecimiento del juez.* El juez, `granite-4.0-1b`, es uno de los doce modelos evaluados y juzga también sus propias respuestas: un juez LLM puede favorecer las suyas [10], sin juez externo ni segundo evaluador que lo controle. Como atenuante, es el modelo con menos respuestas incorrectas involucradas (3 de 32, Tabla 2); no obstante, es un modelo sub-2B que adjudica equivalencia semántica — más exigente que clasificar campos estructurados — sin validación contra anotación humana: los 0 casos de `juez_parse_ok = False` indican salidas parseables, no calidad del criterio, y la métrica laxa es una cota superior de un instrumento no validado.

La versión de la librería como factor de confusión. Como los doce modelos se ejecutan repartidos 9/3 entre los dos grupos de versión de `transformers` (Tabla 1), parte de cualquier diferencia de latencia entre grupos puede deberse a la librería y no al modelo.

Topología de procesadores no controlada. Los dos procesadores lógicos que expone Docker pueden corresponder a un mismo núcleo por *hyperthreading* o a dos distintos, sin que sea observable desde el contenedor (Sección 3.6). Afecta por igual a los doce modelos, sin sesgo relativo, pero los valores absolutos de latencia pueden estar sobreestimados. Los cocientes $\times$`4,10` y $\times$`2,88` comparan pares bajo la misma topología, por lo que son robustos a esta amenaza.

Externa y de conclusión. El dataset es reducido y desbalanceado (Tabla 3), se usó un único prompt sin optimización por modelo, la entrada fue texto limpio sin ruido de reconocimiento automático de habla (ASR) y se cubrió una sola variante dialectal; con esos tamaños de categoría, las diferencias porcentuales entre modelos son indicativas, no estadísticamente significativas. Por último, `attn_implementation` y la `revision` exacta de cada modelo quedaron sin fijar.

7. Conclusiones y trabajo futuro

Este trabajo evaluó automáticamente doce SLMs abiertos de menos de 2 mil millones de parámetros interpretando comandos de domótica en español rioplatense, con un procedimiento de dos etapas y decodificación determinista que reemplaza toda anotación manual y escala sin que el costo de anotación crezca. `granite-4.0-1b` superó en 25,0 puntos porcentuales a la generación anterior de SLMs (Sección 4.1).

La latencia no crece de forma monótona con la exactitud: el modelo más lento del conjunto no es el más exacto, y entre los sub-1B, `Qwen3.5-0.8B` combina la mayor exactitud de su franja con la cuarta latencia más baja de seis (Sección 5). La sobrecarga de la atención híbrida ilustra ese mismo punto — más lenta en los dos tamaños evaluados, pero no sistemáticamente más exacta que su contraparte densa —, lo que sugiere que la elección de un SLM para domótica en el borde debe sopesar arquitectura y generación, no solo el tiempo de cómputo. Ningún modelo del conjunto alcanza una latencia compatible con interacción de voz en tiempo real.

Como trabajo futuro: incorporar un juez externo de mayor escala o validación humana; ampliar y balancear el dataset con voz real transcrita por ASR; cubrir otras variantes dialectales; y repetir el barrido en un host con más de dos procesadores lógicos.

Declaración sobre el uso de IA

Se utilizaron herramientas de inteligencia artificial generativa en el diseño y la curación del dataset y de las taxonomías cerradas, en el desarrollo del código de evaluación, en la redacción del manuscrito y en la generación de figuras y tablas. A esos cuatro roles se agrega un quinto, de naturaleza distinta: un modelo de lenguaje cumple un rol metodológico como juez automático (Sección 3.5). Ese rol no es asistencia editorial, sino parte del método experimental: sus salidas son datos del estudio, sujetos a las mismas amenazas a la validez que cualquier otra medición. Los autores revisaron, validaron y son responsables del contenido, el análisis y las conclusiones, conforme a buenas prácticas de divulgación del uso de agentes de IA en la investigación académica [11].

Referencias

1. Gupta, A., Thomas, B., Asnani, H., et al.: Small Language Models (SLMs) Can Still Pack a Punch: A Survey. arXiv preprint arXiv:2501.05465 (2025)
2. Weld, H., Huang, X., Long, S., Poon, J., Han, S.C.: A Survey of Joint Intent Detection and Slot Filling Models in Natural Language Understanding. ACM Computing Surveys 55(8) (2023)
3. Desot, T., Raimondo, S., Mishakova, A., Portet, F., Vacher, M.: Towards a French Smart-Home Voice Command Corpus: Design and NLU Experiments. In: Text, Speech, and Dialogue (TSD 2018). LNCS, Springer, Heidelberg (2018)
4. Birkmose, R., Reece, N.M., Norvin, E.H., Bjerva, J., Zhang, M.: On-Device LLMs for Home Assistant: Dual Role in Intent Detection and Response Generation. arXiv:2502.12923 (2025)
5. Qwen Team: Qwen2.5 Technical Report. Tech. rep., Alibaba Group (2024)
6. Allal, L.B., et al.: SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model. arXiv:2502.02737 (2025)
7. OLMo Team: 2 OLMo 2 Furious. arXiv preprint arXiv:2501.00656 (2025)
8. Beurer-Kellner, L., Fischer, M., Vechev, M.: Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation. arXiv preprint arXiv:2403.06988 (2024)
9. Dong, Y., Ruan, C.F., et al.: XGrammar: Flexible and Efficient Structured Generation Engine for Large Language Models. arXiv preprint arXiv:2411.15100 (2024)
10. Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E.P., Zhang, H., Gonzalez, J.E., Stoica, I.: Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In: Advances in Neural Information Processing Systems (NeurIPS) (2023)
11. Torassa Colombero, V., Casco, M.E., Roatta, S.E.: Uso ético de los agentes de inteligencia artificial en el ámbito de la investigación académica. In: XXXI Congreso Argentino de Ciencias de la Computación (CACIC 2025) (2025)