FROM FIXED KEYS TO READABLE SCHEMAS: SMALL LANGUAGE MODELS FOR VEHICLE AGENT FUNCTION CALLS

Anonymous authors
Paper under double-blind review

ABSTRACT

In-vehicle assistants are beginning to act on what drivers say, turning a request such as "turn on cruise control" into a call to one of the vehicle's functions. This Vehicle Function Calling must run on the vehicle's own hardware, which calls for small language models (SLMs), and must abstain when no available function fits, because a wrong call acts on hardware. Since the functions differ across vehicles and change with software updates, the key design question is how to represent them to a small model. A Functional Token (FT) model learns one token per function, whereas a Schema-in-Prompt (SIP) model reads the offered functions' schemas in its prompt. Vehicle Function Calling is a new domain: prior vehicle work uses FT on private data, and no public benchmark compares the two. We answer the question in three connected steps. A formal analysis proves that FT can neither call an Unseen Function, one never used as a training target, nor base its abstention on which functions are offered, whereas SIP can do both. Because abstention needs a guarantee, we calibrate it with split conformal risk control. We release a benchmark from the Android Automotive specification and train 10 SLMs under two billion parameters with matched training for each representation. FT scores zero on Unseen Functions, while SIP reaches 15.9 to 76.6%. When a function trained as a target is not offered, only SIP, which reads the offered set, abstains reliably, though it is much slower to its first call. Across model families, size alone does not order Unseen accuracy. The conformal bound holds on random splits, but not when the requested functions shift. How functions are represented, more than model size, decides what a small in-vehicle model can call.

1 INTRODUCTION

In-vehicle assistants are moving from fixed voice commands to language models that turn a driver's request into a call to a vehicle function: "move my seat one step forward" should become a call to the seat-position function with the right arguments, and a request that no available function can serve should end in abstention (Figure 1). Vehicle Function Calling, as we call this task, is a new domain with demands that general tool use does not share. A wrong call actuates hardware; the model has to run in the vehicle under tight memory, latency and connectivity limits, which favors small language models (SLMs); and the functions a vehicle offers change across models, trim levels and software updates. The design question that follows is how to represent the available functions to a small model.

Two representations are in use, and they trade flexibility for cost. A *Functional Token* (FT) model gives each function a dedicated vocabulary token learned in training (Chen & Li, 2026; Sowti Khiabani et al., 2025); its prompt is short, but it can name only the functions it was trained on. A *Schema-in-Prompt* (SIP) model reads the schemas of the offered functions in its input, so it can in principle call a function it never saw as a target, at the price of a much longer prompt. There is currently no evidence on which to choose between them, for three reasons. First, the closest prior study of vehicle function calling with a small model uses eight proprietary functions and releases neither its data nor its code (Sowti Khiabani et al., 2025), and no public vehicle benchmark provides training data or tests calls to functions withheld as training targets. Second, there is no formal account of what each representation can output and observe, so it is unclear which failures training can

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

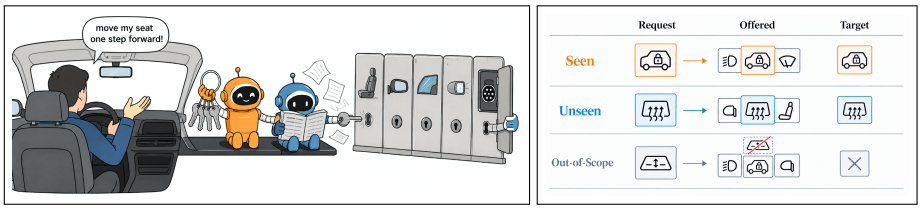


Figure 1: Left: Functional Token gives each function a vocabulary entry, while Schema-in-Prompt reads the offered schemas at inference time. Right: Seen Functions are training targets, Unseen Functions are withheld as targets, and Out-of-Scope Requests have no suitable offered function and require abstention.

fix and which are structural. Third, abstention in a vehicle needs a guarantee, not only a tendency learned in training.

We address all three. We build a benchmark from the Android Automotive `VehiclePropertyIds` specification, whose 79 functions yield Seen Functions used as training targets, Unseen Functions never used as targets, and Out-of-Scope Requests that no offered function can serve. We formalize the task and prove that an FT model can neither call an Unseen Function nor make its abstention depend on which functions are offered, whereas SIP can do both. To give abstention a guarantee, we calibrate the abstention threshold with split conformal risk control, which keeps the expected rate of confident errors, committed calls to a function other than the target, at or below a chosen level α.

We compare the two representations under matched training on 10 SLMs under two billion parameters, including models specialized for function calling, and report three findings. First, the representation decides what a model can call. Both representations are accurate on Seen Functions, but on Unseen Functions FT scores zero by construction while SIP reaches 15.9 to 76.6%, and when a Seen Function is not offered, SIP abstains on most such requests while FT, which is not told what is offered, almost never does; SIP pays for this with a much longer time to first call on the Device Proxy. Second, size alone does not order Unseen accuracy: larger models do better within a family, but a 0.6B model matches or exceeds every larger general-purpose model near 1B, and among the mid-sized models, most of the difference comes from how often they attempt a call rather than from how well they call. Third, the calibrated threshold keeps confident errors at or below α when calibration and test requests are exchangeable, but not when they ask for different functions, so a threshold calibrated on other functions does not transfer.

Our contributions are the following. (i) To our knowledge, the first formal treatment of Vehicle Function Calling, which characterizes Functional Token and Schema-in-Prompt by what each can output and observe (Section 4). (ii) An application of split conformal risk control to abstention, whose guarantee on confident errors we stress-test on the trained models: it holds on random splits and fails when the requested functions shift (Sections 4.4 and 6.3). (iii) To our knowledge, the first public benchmark for training and evaluating small language models on Vehicle Function Calling over a real vehicle specification, and the first vehicle benchmark that tests calls to functions withheld as training targets; we release its data generation, training and evaluation code (Section 3). (iv) A controlled comparison of the two representations in accuracy, abstention and Device Proxy cost (Section 6).

2 Related Work

We review tool use by language models, small models for on-device function calling and calibrated abstention, and end with the vehicle studies closest to ours; Appendix A discusses each in more detail.

Language models can be trained to call external tools (Schick et al., 2023; Patil et al., 2024; Qin et al., 2023), and BFCL also scores requests for which no provided function applies (Patil et al., 2025). Such benchmarks count these requests as one category, whereas in a vehicle the requested function may be absent from the Function Surface or present on it but not offered.

Small on-device models represent the available functions in two ways. Octopus v2 gives each function a dedicated token (Chen & Li, 2026), following ToolkenGPT (Hao et al., 2023), so it cannot name a function it was never trained on; Hammer, xLAM and FunctionGemma read function schemas in the prompt (Lin et al., 2024; Zhang et al., 2024; Google AI for Developers, 2025), and reasoning over tool descriptions reaches unseen tools even without fine-tuning (Wu et al., 2025). To our knowledge, the two representations have not been compared under matched training on one Function Surface, on calls to functions withheld as training targets and on abstention.

Abstention is classification with a reject option (Chow, 1970), and selective classification thresholds a confidence score to trade coverage for risk (Geifman & El-Yaniv, 2017). Conformal risk control calibrates such a threshold to bound the expected loss (Angelopoulos et al., 2024); it has let reasoning models abstain when an answer is likely wrong (Gu et al., 2026) and controlled errors per argument role in tool calls (Rahman et al., 2026). We calibrate one threshold over whole Vehicle Function Calls and test it when the requested functions shift.

Vehicle function calling itself has received little attention. The closest work fine-tunes a pruned language model to call eight proprietary vehicle functions through dedicated tokens and releases neither its data nor its code (Sowti Khiabani et al., 2025), so its results cannot be reproduced or compared with ours. Other recent vehicle studies evaluate large models as agents, in a simulated cockpit (Yang et al., 2025), with a simulated user (Kirmayr et al., 2026), or on whether to execute, decline or confirm a voice command (Afroze et al., 2026). None of them provides training data for small models over a documented Function Surface or tests functions withheld as training targets.

3 Vehicle Function-Calling Benchmark

We release a benchmark for training and evaluating small language models on Vehicle Function Calling, together with the code that generates its data, trains the models and scores them; to our knowledge, no public vehicle benchmark offers training data or tests functions withheld as training targets (Section 2). Its Function Surface $\mathcal{F}$, the set of callable functions, is derived from the Android Automotive `VehiclePropertyIds` specification (Android Open Source Project, 2026): each controllable vehicle property becomes a function schema that keeps the locations where it applies and the values it accepts, so the 79 functions are specified externally rather than invented for the benchmark (Appendix B.1).

Each example pairs a request x with an offered set $O \subseteq \mathcal{F}$, the functions available for that request, and its target is either a Vehicle Function Call to a function in O or abstention, written $\perp$. *Seen Functions* $\mathcal{T} \subseteq \mathcal{F}$ occur as training targets. *Unseen Functions* never do, and related functions are withheld together. Their schemas may still appear in training as non-target candidates, and 15 of the 18 share an argument shape with a training function (Appendix B.4). An *Out-of-Scope Request* is one that no offered function can serve, either because the functionality is absent from $\mathcal{F}$ or because the function exists but is not offered for that example, and its correct answer is abstention. Offered sets mix Seen and Unseen schemas, so their composition reveals neither the split nor the answer (Appendix B.4).

Examples are generated label-first: a valid call is sampled and checked against its schema before a language model writes the request (Appendix B.3). The requests are screened for Target Leakage, where the wording exposes the target's identifier, checked by a second model from a different family, and reviewed by hand in a stratified Verified Slice of 217 examples, of which 100.0% are faithful to their target and 99.1% unambiguous (Appendix B.7). The corpus has 9,822 examples, 6,597 for training and 3,225 for evaluation, over 56 Seen, 18 Unseen and 5 Out-of-Scope functions (Table 2 in Appendix B.6).

4 Mathematical Formalism

Functional Tokens and Schema-in-Prompt differ in two structural ways: in which functions the model can output, and in whether it observes which functions are offered. After defining the task (Section 4.1), we state both differences as theorems (Section 4.2), treat abstention as selection with a reject option (Section 4.3) and calibrate its threshold (Section 4.4); Appendix C gives the extended discussion and the proofs.

4.1 Problem Setup

Let $\mathcal{V}$ be the tokenizer vocabulary, which for FT includes the added Functional Tokens, and let $\mathcal{V}^*$ be the set of finite token sequences over $\mathcal{V}$. The *Function Surface* is a finite set $\mathcal{F} = \{f_1, \ldots, f_m\}$ of callable functions. Each $f \in \mathcal{F}$ has a serialized schema $S(f) \in \mathcal{V}^*$: its identifier, description, arguments and value constraints. An *offered set* $O \subseteq \mathcal{F}$ contains the functions available for a request, $\mathcal{S}(O) \in \mathcal{V}^*$ denotes the serialization of the schemas in O, and $2^{\mathcal{F}}$ denotes the collection of subsets of $\mathcal{F}$.

An *example* is a triple (x, O, y) of a request $x \in \mathcal{V}^*$, an offered set $O \subseteq \mathcal{F}$, and a target $y \in O \cup \{\perp\}$, where $\perp$ denotes abstention; we write (x_i, O_i, y_i) when examples are indexed. Let $\mathcal{C}(x) \subseteq \mathcal{F}$ be the set of functions that can semantically satisfy x. Targets follow the task definition of Section 3:

$$y = \perp \iff \mathcal{C}(x) \cap O = \varnothing, \qquad \text{and otherwise } y \in \mathcal{C}(x) \cap O, \tag{1}$$

where, for a generated in-scope example, y is the function from which the request was generated. An *Out-of-Scope Request* is thus an example with $\mathcal{C}(x) \cap O = \varnothing$: either $\mathcal{C}(x) = \varnothing$, so that the requested functionality is absent from the Function Surface, or $\mathcal{C}(x) \neq \varnothing$ but no capable function is offered. Let $\mathcal{T} \subseteq \mathcal{F}$ be the Seen Functions, the only functions that occur as training targets and, under FT, the only functions with a Functional Token. Outputs are compared at the level of function identity; exact-call accuracy additionally requires correct arguments, so impossibility statements below hold a fortiori for exact accuracy.

We write $p(z \mid \cdot)$ for the distribution of the adapted FT or SIP model over generated sequences $z \in \mathcal{V}^*$; generation ends at an end-of-sequence token or after a fixed maximum number of tokens, the *generation limit*. A decoder maps a generated sequence to the function it calls, to $\perp$ if it abstains, and to the *invalid* outcome $\diamond$ otherwise: $\mathrm{dec}_{\mathrm{SIP}}$ reads the function from its identifier and can return any function of $\mathcal{F}$, whereas $\mathrm{dec}_{\mathrm{FT}}$ reads it from its Functional Token and can return only functions of $\mathcal{T}$. The task-level distributions $p_{\mathrm{SIP}}(y \mid x, \mathcal{S}(O))$ and $p_{\mathrm{FT}}(y \mid x)$ are the probabilities that a generated sequence decodes to $y \in \mathcal{F} \cup \{\perp, \diamond\}$ (Appendix C.1).

We call $\mathcal{Y}_{\mathrm{SIP}}(O) = O \cup \{\perp\}$ the *output support* of SIP; the correct output of an example always lies in it. FT can output only Seen Functions, $\perp$ or the invalid $\diamond$, which is never correct, so it can be correct on an example only if the target lies in $(\mathcal{T} \cap O) \cup \{\perp\}$. A *decision rule* maps a request and an offered set to an outcome, $\hat{y} : \mathcal{V}^* \times 2^{\mathcal{F}} \to \mathcal{F} \cup \{\perp\}$, and is correct on (x, O, y) if $\hat{y}(x, O) = y$; an FT rule ignores its second argument, because the FT input contains no schemas.

4.2 Schema-in-Prompt versus Functional Token Representation

The first structural difference is which functions each decoder can return.

Theorem 4.1 (Output support). *Fix a request $x \in \mathcal{V}^*$ and an offered set $O \subseteq \mathcal{F}$.*

- *(a) For every outcome $y \in \mathcal{F} \cup \{\perp\}$ and every sequence z that decodes to y under $\mathrm{dec}_{\mathrm{SIP}}$, $p_{\mathrm{SIP}}(y \mid x, \mathcal{S}(O)) \geq p(z \mid x, \mathcal{S}(O))$. Hence $p_{\mathrm{SIP}}(y \mid x, \mathcal{S}(O)) > 0$ if and only if some sequence that decodes to y has positive probability under the SIP model.*
- *(b) If every next-token distribution of the SIP model assigns positive probability to every token, then every sequence the model can emit within the generation limit has positive probability, so $p_{\mathrm{SIP}}(y \mid x, \mathcal{S}(O)) > 0$ for every outcome $y \in \mathcal{F} \cup \{\perp\}$ to which such a sequence decodes; in particular this holds for every such function, whether it is Seen or Unseen, offered or not.*
- *(c) For every $f \in \mathcal{F} \setminus \mathcal{T}$ and every request $x' \in \mathcal{V}^*$, $p_{\mathrm{FT}}(f \mid x') = 0$, whatever the FT parameters.*

Theorem 4.1 is a statement about output spaces, not about learning, and it holds for every parameter value; Appendix C.2 gives the proof. Its substance is the contrast between (b) and (c): a softmax SIP model sampled without truncation gives positive probability to every offered function, Seen or Unseen, whereas the FT decoder cannot return a function without a trained token. Part (c) holds by construction, also under greedy decoding, and predicts that FT's exact Unseen accuracy is zero for every model and seed and that every FT call on an Unseen request is an error (Table 1, and Table 6 in Appendix D.3).

The second structural difference is what each representation observes: the FT input contains only the request, so an FT rule cannot distinguish two examples that share a request but differ in what is offered, see Theorem 4.2.

Theorem 4.2 (Conditioning on the offered set)**.** *Let $\hat{y}$ be a decision rule that depends on the request alone, $\hat{y}(x, O) = \hat{y}(x, O')$ for all x, O, O', and write $\hat{y}(x)$ for this common value; every FT rule is of this kind.*

(a) If two examples (x, O, y) and (x, O', y') share the request x and satisfy $\mathcal{C}(x) \cap O \neq \varnothing = \mathcal{C}(x) \cap O'$, then $\hat{y}$ is incorrect on at least one of them.

(b) For an example (x, O, y) drawn from any distribution over examples, $\mathbb{P}\big(\hat{y}(x) \neq y\big) \geq \mathbb{E}\big[\min\{\mathbb{P}(y = \perp \mid x),\, \mathbb{P}(y \neq \perp \mid x)\}\big]$, where the expectation is over the request x.

Theorem 4.2 says that a rule which does not see the offered set gets at least one of two examples wrong whenever they share a request but differ in whether a capable function is offered, and part (b) turns this into a lower bound on its error rate. It predicts where Functional Token abstention fails: on Out-of-Scope Requests whose function has a Functional Token but is not offered. Schema-in-Prompt observes the offered set and is not bound by it; Appendix C.3 gives the proof.

SIP pays for observing O in prompt length: its prompt adds the serialized schemas of every offered function to the request, which costs prefill work and cache memory (Appendix C.1).

4.3 Abstention

The correct output is the same for both representations, since it is fixed by equation 1; what differs is the information each decision rule can use. To analyze selection and abstention together, we write SIP's decision as selection with a reject option. Let $s(x, f) \in \mathbb{R}$ be a compatibility score between a request x and a function f, for instance the log-probability that the SIP model calls f. In an SIP model the score of f generally depends on the whole prompt, and hence also on O; we allow this and suppress O from the notation, except under Assumption C.1 (Appendix C.5), which rules it out. Let $\gamma(x, O) \in [-\infty, +\infty]$ be a possibly context-dependent rejection threshold. The thresholded rule is

$$\hat{y}_{\mathrm{SIP},\gamma}(x, O) = \begin{cases} \arg\max\limits_{f \in O} s(x, f), & \max_{f \in O} s(x, f) > \gamma(x, O), \\ \perp, & \text{otherwise.} \end{cases} \tag{2}$$

Ties in the $\arg\max$ are broken by a fixed order on $\mathcal{F}$, in favor of $\perp$ against a function (Appendix C.1); the rule abstains at ties with the threshold and, with the convention $\max_{f \in \varnothing}(\cdot) = -\infty$, whenever $O = \varnothing$; its output always lies in $\mathcal{Y}_{\mathrm{SIP}}(O)$. With a constant threshold it is Chow's reject rule (Chow, 1970) and the confidence-thresholding rule of selective classification (Geifman & El-Yaniv, 2017), with abstention as an additional output.

Abstention need not be modeled separately from selection. Appendix C.4 shows that ordinary supervised training learns abstention as one more class, and that the mode of a trained SIP model is itself an instance of rule equation 2, with the log-probability as $s(x, f)$ to implement the threshold on.

Theorem 4.1 shows that SIP *can* select an Unseen Function, not that it will; Theorem C.2 in Appendix C.5 describes how SIP errs on an Unseen target whose schema is close to that of another offered function.

4.4 Risk Control

The threshold in equation 2 has so far been arbitrary. For deployment, the quantity to control is the rate of *confident errors*: executing a function that is not the target, which covers both over-triggering on Out-of-Scope Requests and wrong calls on in-scope ones. Split conformal risk control (Angelopoulos et al., 2024) calibrates a constant threshold on held-out examples so that this rate is at most a chosen α in expectation. The result below is a direct instance of their Theorem 1; we state it, with a self-contained proof, because the loss and the conditions on the score are specific to our setting.

For a constant $\gamma \in [-\infty, +\infty]$, write $\hat{y}_{\mathrm{SIP},\gamma}$ for rule equation 2 with $\gamma(x, O) \equiv \gamma$; at $\gamma = +\infty$ it always abstains, and at $\gamma = -\infty$ it commits whenever $O \neq \varnothing$. Raising γ can only turn a committed prediction into an abstention, never the reverse, and it never changes the committed function. Given $n \geq 1$ calibration examples (x_i, O_i, y_i), $i = 1, \ldots, n$, define, with $\mathbf{1}[\cdot]$ the indicator function, the per-example loss and the empirical risk

$$L_i(\gamma) = \mathbf{1}\big[\hat{y}_{\mathrm{SIP},\gamma}(x_i, O_i) \notin \{\perp, y_i\}\big], \qquad \hat{R}(\gamma) = \frac{1}{n}\sum_{i=1}^{n} L_i(\gamma),$$

and, for a target rate $\alpha \in (0, 1)$, the calibrated threshold

$$\hat{\gamma} = \inf\left\{\gamma \in [-\infty, +\infty] \,\middle|\, \hat{R}(\gamma) \leq \alpha - \frac{1-\alpha}{n}\right\}, \qquad \inf \varnothing := +\infty,$$

which depends only on the calibration examples.

Theorem 4.3 (Error control)**.** *Let examples $1, \ldots, n$ be the calibration examples and example $n+1$ a test example $(x_{n+1}, O_{n+1}, y_{n+1})$. Suppose that the score s, which may depend on the offered set, and the tie-breaking order are fixed, i.e. do not depend on these $n+1$ examples; for a learned score this means that it is trained on data independent of them, and the statement then holds conditionally on the training data. Suppose also that the $n+1$ examples are exchangeable, as they are when drawn i.i.d. from one distribution over examples. Then, for every $\alpha \in (0, 1)$,*

$$\mathbb{P}\big(\hat{y}_{\mathrm{SIP},\hat{\gamma}}(x_{n+1}, O_{n+1}) \notin \{\perp, y_{n+1}\}\big) \leq \alpha,$$

where $\mathbb{P}$ is over the joint draw of the $n+1$ examples. Moreover, $\hat{\gamma} < +\infty$ if and only if $\alpha \geq 1/(n+1)$; if $\alpha < 1/(n+1)$, the calibrated rule abstains on every example.

Theorem 4.3 says that a threshold calibrated on n held-out examples keeps the probability of committing to a wrong output at most α on a fresh example exchangeable with them, whatever the score, provided the score does not depend on those examples. The guarantee is marginal over the mix of Seen, Unseen and Out-of-Scope Requests and needs calibration examples exchangeable with deployment ones; Section 6.3 tests both properties, and Appendix C.6 states its caveats and gives the proof. The proof uses rule equation 2 only through two facts, that the rule commits exactly when a score exceeds γ and that the answer it commits to does not depend on γ, so the theorem holds for any rule that commits to a fixed answer whenever a fixed score exceeds the threshold, with any committed answer other than the target counted as a confident error. The rule calibrated in Section 6.3, which commits the greedy completion of the call branch when its sequence log-odds against abstention exceed the threshold, is of this kind.

5 Experimental Setup

We fine-tune 10 open-weight models, from Gemma 3-270M to Qwen3-1.7B, with both representations, on the same training examples and the same full fine-tuning recipe, giving 20 cells of 5 independently trained seeds each (Appendix D.1). The models form a matched pair that differs only in function-calling specialization (FunctionGemma-270M and Gemma 3-270M), a step in size within Gemma 3 and within Qwen3, and three general-purpose families near 1B, with size counted in non-embedding parameters. Three function-calling models derived from Qwen2.5 (Hammer2.1-0.5B, Arch-Function-1.5B and xLAM-2-1B) needed a guard against non-finite gradients; since specialization, Qwen2.5 lineage and the guard vary together in them, no claim about specialization rests on them. We report exact-match accuracy on Seen and Unseen Functions and correct-abstention accuracy on Out-of-Scope Requests, and factor Unseen accuracy into the *attempt rate*, the fraction of requests on which a model calls any function, and the *accuracy given an attempt*. A paired difference between two cells is *separable* when the 95% intervals of both bootstrap schemes exclude zero, *unresolved* when only the primary one does, and *not separable* otherwise; the intervals resample only the few held-out property groups, so they are approximate, and we do not correct for multiple comparisons (Appendix D.2). Latency is measured on a Device Proxy, a server CPU running each cell's 4-bit export, and makes no claim about a vehicle (Appendix D.10).

Table 1: Accuracy (%) of every main-grid cell; Out-of-Scope accuracy is correct abstention. Mean accuracy in percent, row-weighted over each split's rows and averaged over 5 seeds per row. [c]0.0 by construction: a Functional Token model has no token for a function it was not trained on, so it cannot call one. [g]Trained with a non-finite gradient guard, which skips any optimiser step whose gradient is not finite; it discarded 3 to 11 of 1,236 steps per run, so these models received at least 99.1% of the updates the others did.

	Functional Token			Schema-in-Prompt		
Model	Seen ↑	Unseen ↑	Out-of-Scope ↑	Seen ↑	Unseen ↑	Out-of-Scope ↑
Gemma 3-270M	97.4	0.0[c]	81.1	97.5	15.9	96.6
FunctionGemma-270M	97.0	0.0[c]	81.1	97.1	17.9	97.1
Hammer2.1-0.5B[g]	97.7	0.0[c]	82.4	98.8	46.9	97.9
Qwen3-0.6B	97.5	0.0[c]	82.6	98.8	61.0	98.3
Gemma 3-1B	98.0	0.0[c]	83.0	99.2	29.4	98.1
Llama-3.2-1B	98.1	0.0[c]	82.6	99.2	43.9	98.4
OLMo-2-1B	97.6	0.0[c]	83.3	98.9	44.9	98.6
Arch-Function-1.5B[g]	98.1	0.0[c]	83.0	99.1	73.7	98.6
xLAM-2-1B[g]	97.5	0.0[c]	82.8	98.7	71.0	98.5
Qwen3-1.7B	97.4	0.0[c]	82.7	99.0	76.6	98.5

6 Numerical Results

We first compare the two representations on what they can call and when they abstain (Section 6.1), and then ask what Schema-in-Prompt's Unseen accuracy depends on (Section 6.2). We then test whether abstention can be calibrated (Section 6.3) and what reading schemas costs (Section 6.4).

6.1 Representation Decides What Can Be Called

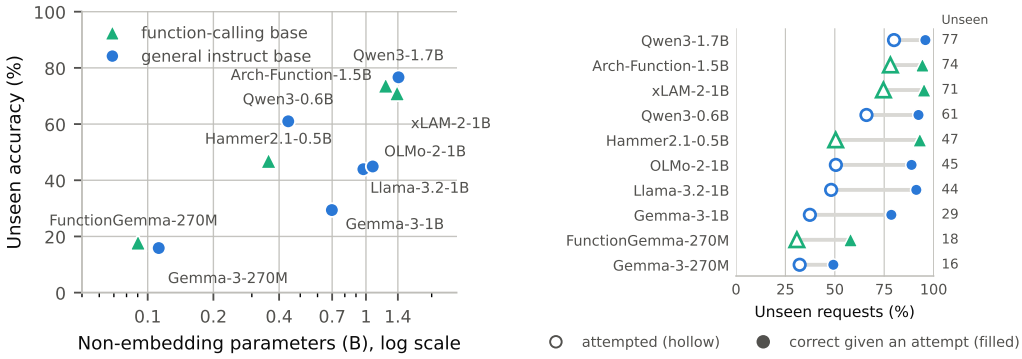
On Seen Functions the representation barely matters, with accuracy of 97.0 to 98.1% for Functional Token and 97.1 to 99.2% for Schema-in-Prompt at every size (Table 1); on Unseen Functions, it decides whether the target can be called at all. Every Functional Token cell scores zero by construction, because the model has no token for a function that was never a training target (Theorem 4.1(c)); its few calls on Unseen requests, 0.4 to 1.7% of them, all execute a function the driver did not ask for (Table 6 in Appendix D.3). Schema-in-Prompt reaches 15.9 to 76.6%, above zero for every model, because the target's schema is part of its input.

Out-of-Scope Requests expose the second structural difference. Schema-in-Prompt cells abstain on 96.6 to 98.6% of them and Functional Token cells on 81.1 to 83.3%, and the gap lies in requests whose function exists on the Function Surface but is not offered (Appendix D.4, Figure 7): when that function has a Functional Token, Functional Token cells abstain on only 2.0 to 4.6% of them, against 84.6 to 95.0% for Schema-in-Prompt, and when it has none, both abstain on almost all of them (98.8 to 100.0% and 98.8 to 99.6%). As Theorem 4.2 predicts, a Functional Token model sees only the request, so it cannot decline a function it knows on the grounds that the function is unavailable. Restricting its decoding to the tokens of offered functions would remove this obstruction (Appendix C.3); we compare Functional Token as trained, on the request alone.

6.2 Size Alone Does Not Order Unseen Accuracy

Within Schema-in-Prompt, Unseen accuracy tends to rise with size but is not ordered by parameter count across families (Figure 2a). Qwen3-0.6B, with 0.44B non-embedding parameters, has a higher point estimate than every larger general-purpose model near 1B and is separably ahead of Gemma 3-1B (0.70B), by $+21.8$ points $[+8.9, +36.7]$; its margins over Llama-3.2-1B ($+10.8$) and OLMo-2-1B ($+8.4$) are unresolved.

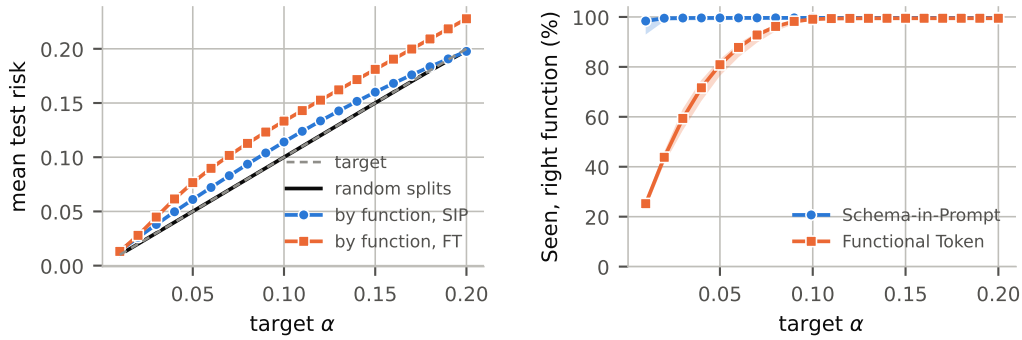
Size helps within a family, but the family matters about as much: the step from Gemma 3-270M to Gemma 3-1B adds $+18.5$ points $[+4.9, +34.4]$, the step from Qwen3-0.6B to Qwen3-1.7B adds $+12.3$ $[+6.3, +19.2]$, and OLMo-2-1B exceeds Gemma 3-1B, a general-purpose model of similar size, by $+13.4$ $[+4.6, +22.8]$. The design cannot separate a family's effect from how much function calling its base model saw before fine-tuning.



(a) Unseen accuracy against model size.

(b) Attempt rate and accuracy given an attempt.

Figure 2: Schema-in-Prompt on Unseen Functions. (a) Accuracy against non-embedding parameters. (b) The same accuracy factored into attempt rate (hollow) and accuracy given an attempt (filled), with Unseen accuracy at the right. Triangles mark function-calling base models, circles general-purpose ones.



(a) Mean test risk against the target α.

(b) Seen right-function rate against α.

Figure 3: Conformal risk control of abstention (Theorem 4.3). (a) Confident-error risk on random splits (black, pooled over cells) and on splits by function (per representation); dashed, risk $= \alpha$. (b) Both representations calibrated on Seen and Out-of-Scope requests; means over the 10 models, bands their range.

Specialization for function calling can be isolated only at 270M, where FunctionGemma-270M and Gemma 3-270M differ by $+2.7$ points $[-2.2, +10.0]$, which is not separable. At 0.5B, the guarded Hammer2.1 minus the general-purpose Qwen3-0.6B is -9.7 points (unresolved).

Of the two ablations in Appendix D.7, LoRA is not separable from full fine-tuning on Unseen, and FunctionGemma's native format raises its Unseen and lowers its Out-of-Scope accuracy.

Much of the spread in Unseen accuracy comes from how often a model attempts a call rather than from how well it calls (Figure 2b). Across Qwen3-0.6B, Llama-3.2-1B and OLMo-2-1B, the attempt rate spans 17.9 points, while accuracy given an attempt spans only 3.5, and the 270M models are weaker on both (Appendix D.8); Schema-in-Prompt's Unseen errors are thus mostly abstentions. The attempt rate also reflects where the abstention threshold of rule equation 2 sits, so a ranking by pooled Unseen accuracy is not a ranking of generalization.

6.3 Calibrating Abstention

We test Theorem 4.3 on all 20 cells and 5 seeds, scoring each example by the log-odds between the greedy completion of the call branch and the abstention, each as a complete sequence, so that the committed call does not depend on the threshold. Calibration and test sets are drawn from the evaluation requests (Appendix D.9), so α is a budget over their mix of Seen, Unseen and Out-of-Scope Requests, not over any deployment.

Figure 3a shows where the guarantee holds. On random splits, the calibrated threshold keeps the mean test risk at the target α or below in every cell, so the black line stays at or below the dashed diagonal. Because the loss is function-level, it ignores argument errors: under it, the exact-call error of the worst Schema-in-Prompt cell reaches 11.1% at $\alpha = 0.05$, while calibrating on the exact-call loss keeps it at or below 5.0% (Appendix D.9). When calibration and test sets ask for different functions and request types, the risk exceeds the target for both representations: at $\alpha = 0.05$ it is 5.7 to 7.0% for Schema-in-Prompt and 7.0 to 8.6% for Functional Token (Table 14 in Appendix D.9). Exchangeability must therefore extend to which functions are requested.

Calibration moves each model toward its budget (Table 15 in Appendix D.9). A Schema-in-Prompt model whose own error rate is below α commits on more requests, raising both its right-function rate on Unseen requests and its Out-of-Scope commit rate (up to 22.9% at $\alpha = 0.05$), while one above α, such as the 270M models, abstains more. Calibration therefore trades confident errors against Unseen coverage: for a model already below α, it raises confident errors up to the budget.

When both representations are calibrated on Seen and Out-of-Scope requests only (on Unseen requests Functional Token is wrong whenever it commits), they differ as Theorem 4.2 predicts (Figure 3b). At $\alpha = 0.02$, Functional Token calls the right function on 41.3 to 46.1% of Seen requests, against 98.3 to 100.0% for Schema-in-Prompt, in every model (Table 16 in Appendix D.9). Functional Token can over-trigger less only by also abstaining on Seen requests that read the same.

Calibrating without Unseen requests and testing on all requests breaks the bound for every cell except the three strongest Schema-in-Prompt models (Table 17 in Appendix D.9). A threshold calibrated on other functions therefore does not transfer in general, and the remedy is to recalibrate when the requested functions change.

6.4 The Cost of Reading Schemas

Reading schemas costs prompt length. A Schema-in-Prompt prompt carries 1,784 to 1,921 tokens against 14 to 16 for Functional Token, and its prefill takes 122 to 154 times the floating-point operations, a count that does not depend on hardware. On the Device Proxy, this becomes a ratio of 19.7 to 39.5 in time to first call between the two representations of each model, with Schema-in-Prompt taking from 2277 ms to 10587 ms (Appendix D.10).

When the offered set does not change, caching the processed schema prefix shortens Schema-in-Prompt's Device Proxy time to first call by a factor of 4.7 to 12.2 and changes no accuracy by more than 0.6 points over the seeds measured (Appendix D.10). The cache must be rebuilt whenever the offered functions change.

The 4-bit exports that the Device Proxy times can score well below the 16-bit checkpoints: the 270M Functional Token exports lose 14.0 to 18.6 points on Seen Functions (Appendix D.11).

7 Conclusion

An in-vehicle assistant needs a small on-device model that turns a driver's request into the right Vehicle Function Call, or abstains, even as the vehicle's functions change. How these functions are represented determines much of what such a model can do: a Functional Token model cannot call a function it was not trained on, nor tell whether a function it knows is offered, and these limits are structural. A Schema-in-Prompt model is bound by neither, at the price of a much slower first call. The benchmark we release confirmed these predictions across 10 models under matched training. It also showed that, across model families, size alone does not order Unseen accuracy, and that the conformal bound on confident errors holds on random splits but not when the requested functions shift. Functional Tokens suit a fixed surface whose trained functions are always offered and a tight latency budget; Schema-in-Prompt suits a surface that changes after training, with abstention recalibrated when the requested functions change. Our requests are synthetic and English, so natural driver requests, other languages, Compound Requests, Multi-Turn Agentic Tasks and in-car measurements are the next steps. Because a vehicle's functions keep changing after its model ships, we expect the choice of representation, together with a calibrated rule for when to abstain, to matter as much as the choice of model.

AI USE STATEMENT

We used generative AI tools to generate the synthetic dataset, to implement methods, to give feedback on the experimental design, to refine the formulation of our theoretical results and assist in writing their proofs, and to help interpret results. Translation and qualitative data analysis are not applicable to this work. We also used generative AI tools to draft and edit parts of the paper, to search for and summarize related literature, to write the scripts that draw the figures and tables, and to create the artwork in Figure 1. Each use was checked independently of the tool that produced it. Within the method, a language model writes each benchmark request only after its target call has been generated and validated programmatically. The requests are then screened for Target Leakage, checked by a model from a different family, and reviewed by hand in a Verified Slice (Section 3 and Appendix B). AI-assisted code is covered by automated tests, every reported number and figure is regenerated by script from logged predictions, and the authors verified every proof. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies models that would act on a vehicle's functions, but it evaluates them offline on a benchmark and never on a vehicle, so a deployed system would need safeguards that this work does not evaluate, such as confirmation before safety-relevant actions. The benchmark is synthetic and English, so it may not reflect how drivers actually speak. The work involves no human subjects and no personal data, since the only human annotation is the authors' review of the Verified Slice and of the abstention targets the screen flagged, and every specification and model is used under its license.

REPRODUCIBILITY STATEMENT

We release the code for data generation, training and evaluation, together with the scripts that regenerate every table and figure, as anonymous supplementary material. Section 3 and Appendix B describe how the benchmark is built from the public Android Automotive specification, Appendix C gives complete proofs of the theoretical results, and Appendix D gives the models, the training and inference settings, the evaluation metrics and statistical tests, and the Device Proxy measurements. Because GPU training is not bitwise reproducible, we report every main-grid result over 5 independently trained seeds, and give the seed count of every other result where it is reported.

REFERENCES

Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matthew Stallone, Rameswar Panda, Yara Rizk, GP Bhargav, Maxwell Crouse, Chulaka Gunasekara, Shajith Ikbal, Sachin Joshi, Hima Karanam, Vineet Kumar, Asim Munawar, Sumit Neelam, Dinesh Raghu, Udit Sharma, Adriana Meza Soria, Dheeraj Sreedhar, Praveen Venkateswaran, Merve Unuvar, David Cox, Salim Roukos, Luis Lastras, and Pavan Kapanipathi. Granite-function calling model: Introducing function calling abilities via multi-task learning of granular tasks, 2024. URL https://arxiv.org/abs/2407.00121.

Diba Afroze, Xingli Zhang, Yazhou Tu, and Xiali Hei. From intent to action: Benchmarking LLM safety in vehicle voice command authorization, 2026. URL https://arxiv.org/abs/2609.19630.

Android Open Source Project. VehiclePropertyIds. Android Automotive API reference, 2026. URL https://developer.android.com/reference/android/car/VehiclePropertyIds. Accessed 2026-08-24.

Anastasios N. Angelopoulos, Stephen Bates, Adam Fisch, Lihua Lei, and Tal Schuster. Conformal risk control. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2208.02814.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, Wulong Liu, Xinzhi Wang, Defu Lian, Baoqun Yin, Yasheng Wang, and Wu Liu. ACEBench: Who wins the match point in tool usage?, 2025. URL https://arxiv.org/abs/2501.12851.

Wei Chen and Zhiyuan Li. Octopus v2: On-device language model for super agent, 2026. URL https://arxiv.org/abs/2404.01744.

C. K. Chow. On optimum recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970. doi: 10.1109/TIT.1970.1054406. URL https://ieeexplore.ieee.org/document/1054406.

Can Cui, Zichong Yang, Yupeng Zhou, Yunsheng Ma, Juanwu Lu, Lingxi Li, Yaobin Chen, Jitesh Panchal, and Ziran Wang. Personalized autonomous driving with large language models: Field experiments, 2023. URL https://arxiv.org/abs/2312.09397.

Thierry Deruyttere, Simon Vandenhende, Dusan Grujicic, Luc Van Gool, and Marie-Francine Moens. Talk2Car: Taking control of your self-driving car. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, 2019. doi: 10.18653/v1/D19-1215. URL https://aclanthology.org/D19-1215/.

Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Tinyagent: Function calling at the edge, 2024. URL https://arxiv.org/abs/2409.00608.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, 2017. URL https://papers.nips.cc/paper/7073-selective-classification-for-deep-neural-networks.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems*, 2024. URL https://arxiv.org/abs/2311.04934.

Google AI for Developers. FunctionGemma model overview. Google AI for Developers documentation, 2025. URL https://ai.google.dev/gemma/docs/functiongemma. Accessed 2026-08-27.

Yu Gu, Zijun Yu, Vahid Partovi Nia, and Masoud Asgharian. Pause and reflect: Conformal aggregation for chain-of-thought reasoning. In *Advances in Neural Information Processing Systems*, 2026. URL https://arxiv.org/abs/2605.14098.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. ToolkenGPT: Augmenting frozen language models with massive tools via tool embeddings. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.11554.

Johannes Kirmayr, Lukas Stappen, and Elisabeth André. CAR-bench: Evaluating the consistency and limit-awareness of LLM agents under real-world uncertainty, 2026. URL https://arxiv.org/abs/2601.22027.

Varun Kotte. When can conformal risk control certify LLM outputs? bounds, impossibility, and adaptation for structured generation, 2026. URL https://arxiv.org/abs/2606.29054.

Chenghao Li, Liu Liu, Baosheng Yu, Jiayan Qiu, and Yibing Zhan. Re-initialization token learning for tool-augmented large language models, 2025. URL https://arxiv.org/abs/2506.14248.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://aclanthology.org/2023.emnlp-main.187/.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. Hammer: Robust function-calling for on-device language models via function masking, 2024. URL https://arxiv.org/abs/2410.04587.

Hiroki Matsutani, Naoki Matsuda, and Naoto Sugiura. Accelerating local LLMs on resource-constrained edge devices via distributed prompt caching, 2026. URL https://arxiv.org/abs/2602.22812.

Jeffery Opoku and David Banahene. ToolChain-CRC: Conformal risk control for agentic AI under retrieval and tool-use drift, 2026. URL https://arxiv.org/abs/2606.18467.

Lingyou Pang, Lei Huang, Jianyu Lin, Tianyu Wang, Alexander Aue, and Carey E. Priebe. Taming variability: Randomized and bootstrapped conformal risk control for LLMs, 2025. URL https://arxiv.org/abs/2509.23007.

Varatheepan Paramanayakam, Andreas Karatzas, Iraklis Anagnostopoulos, and Dimitrios Stamoulis. Less is more: Optimizing function calling for LLM execution on edge devices, 2024. URL https://arxiv.org/abs/2411.15399.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2305.15334.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world APIs, 2023. URL https://arxiv.org/abs/2307.16789.

Md Ashikur Rahman, Md Arifur Rahman, Niamul Hassan Samin, Khandaker Rifah Tasnia, Md Hasibul Amin, Sifat Rahman Ahona, and Juena Ahmed Noshin. Beyond aggregate risk: Role-stratified conformal risk control for LLM tool calls, 2026. URL https://arxiv.org/abs/2607.24343.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. DriveLM: Driving with graph visual question answering. In *European Conference on Computer Vision*, 2024. URL https://arxiv.org/abs/2312.14150.

Yahya Sowti Khiabani, Farris Atif, Chieh Hsu, Sven Stahlmann, Tobias Michels, Sebastian Kramer, Benedikt Heidrich, M. Saquib Sarfraz, Julian Merten, and Faezeh Tafazzoli. Optimizing small language models for in-vehicle function-calling, 2025. URL https://arxiv.org/abs/2501.02342.

Sina Tayebati, Divake Kumar, Nastaran Darabi, Dinithi Jayasuriya, Ranganath Krishnan, and Amit Ranjan Trivedi. Learning conformal abstention policies for adaptive risk management in large language and vision-language models, 2025. URL https://arxiv.org/abs/2502.06884.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.,

2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Jun Wang, Jiamu Zhou, Muning Wen, Xiaoyun Mo, Haoyu Zhang, Qiqiang Lin, Cheng Jin, Xihuai Wang, Weinan Zhang, Qiuying Peng, and Jun Wang. HammerBench: Fine-grained function-calling evaluation in real mobile assistant scenarios. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025. URL https://aclanthology.org/2025.findings-acl.175/.

Mengsong Wu, Tong Zhu, Han Han, Xiang Zhang, Wenbiao Shao, and Wenliang Chen. Chain-of-tools: Utilizing massive unseen tools in the CoT reasoning of frozen language models, 2025. URL https://arxiv.org/abs/2503.16779.

Jie Yang, Jiajun Chen, Zhangyue Yin, Shuo Chen, Yuxin Wang, Yiran Guo, Yuan Li, Yining Zheng, Xuanjing Huang, and Xipeng Qiu. VehicleWorld: A highly integrated multi-device environment for intelligent vehicle interaction, 2025. URL https://arxiv.org/abs/2509.06736.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Awalgaonkar, Rithesh Murthy, Eric Hu, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xlam: A family of large action models to empower ai agent systems, 2024. URL https://arxiv.org/abs/2409.03215.

Lucen Zhong, Zhengxiao Du, Xiaohan Zhang, Haiyi Hu, and Jie Tang. ComplexFuncBench: Exploring multi-step and constrained function calling under long-context scenario, 2025. URL https://arxiv.org/abs/2501.10132.

A Extended Related Work

This appendix extends each paragraph of Section 2 with work the section does not cite, and adds work on prompt caching.

Beyond the benchmarks discussed there, API-Bank evaluates calls over a large API collection (Li et al., 2023); ACEBench, ComplexFuncBench and HammerBench add long contexts and multi-step calls (Chen et al., 2025; Zhong et al., 2025; Wang et al., 2025); and τ-bench and τ^2-bench evaluate agents that interact with a simulated user (Yao et al., 2024; Barres et al., 2025). Granite's function-calling model trains relevance detection as a task of its own (Abdelaziz et al., 2024). Our Out-of-Scope split refines the category in which no function applies by the reason it applies to none, since a function withheld from the offered set and a function absent from the Function Surface call for the same abstention but test different information.

Among small function-calling models, TinyAgent and "Less is More" shorten the prompt by selecting only the tools relevant to a request (Erdogan et al., 2024; Paramanayakam et al., 2024), which reduces the cost that Schema-in-Prompt pays for reading schemas. On the learned-token side, initializing a new tool token from the tool's description shortens, but does not remove, the training a new token needs (Li et al., 2025), so a Functional Token model still cannot call a function it was never trained on (Theorem 4.1).

Other conformal methods for language models learn abstention thresholds on top of conformal prediction (Tayebati et al., 2025), reduce the variability of conformal risk control across calibration draws (Pang et al., 2025), which our experiment shows as the spread over 2,000 splits, control risk over multi-step agent trajectories under retrieval and tool drift (Opoku & Banahene, 2026), and delimit what conformal risk control can certify for structured outputs (Kotte, 2026). We calibrate a single threshold over whole calls and report where its guarantee holds and where it fails.

Schema-in-Prompt pays for its flexibility in prompt length, and caching the attention states of a fixed prompt segment is the standard way to reduce that cost. Prompt Cache reuses precomputed attention for recurring segments such as schemas and system messages (Gim et al., 2024), and distributed prompt caching extends the idea to cooperating edge devices (Matsutani et al., 2026). Our cached-schema measurements (Appendix D.10) apply the same idea to the offered schemas on the Device Proxy; we did not find work that studies schema caching on in-vehicle hardware.

Language interfaces for vehicles have mostly addressed perception and driving: grounding passenger commands in street scenes (Deruyttere et al., 2019), question answering for driving (Sima et al., 2024), and translating commands into driving controls (Cui et al., 2023). None of them calls a vehicle's functions. Among the function-calling studies discussed in Section 2, the simulated cockpit environment (Yang et al., 2025) is closest to our representation question: it finds that predicting a target system state works better than issuing stateless calls, which concerns how system state is represented rather than how the Function Surface is, and it evaluates multi-turn interaction.

B Benchmark Construction and Validation

This appendix details how the benchmark of Section 3 is built and checked: the Function Surface, the label-first generation of examples, the offered sets and Out-of-Scope Requests, the training and evaluation partition, and quality control.

B.1 Function Surface Construction

The Function Surface is derived from Android Automotive `VehiclePropertyIds` (Android Open Source Project, 2026). We keep the properties that control the vehicle, drop those that only report its state, and convert each kept control into a callable schema using the constraints of the documented interface. A control that applies to several locations, such as different seats or windows, takes the location as an argument, and documented ranges, units, Boolean states and enumerated modes fix the admissible values of its other arguments. The surface is fixed before any example is generated, and data construction and evaluation use the same definitions.

The 79 functions fall into three groups. Seen and Unseen Functions can be the targets of a Vehicle Function Call, and the 18 Unseen Functions are never training targets. The Out-of-Scope group

Seen	Request: *"Lock the boot."* Label: `door_set_lock(area="DOOR_REAR", on=true)` Offered: `seat_move_lumbar_vertical`, `mirror_set_heat`, `rear_fog_lights_set`, `front_fog_lights_set`, `seat_set_ventilation`, `seat_move_lumbar_side`, `hvac_set_defroster`, `door_set_lock`, `hvac_set_max_defrost`, `ev_set_charge_limit`, `seat_move_cushion_side_support`, `seat_move_depth`, `steering_wheel_set_heat`, `turn_signal_set` The target function was a training target. `0-003954`
Unseen	Request: *"Turn on the rear demister."* Label: `hvac_set_defroster(area="WINDOW_REAR_WINDSHIELD", on=true)` Offered: `seat_move_lumbar_side`, `rear_fog_lights_set`, `headlights_set`, `front_fog_lights_set`, `ev_set_charging`, `adas_set_blind_spot_warning`, `seat_move_headrest_height`, `hvac_set_fan_direction`, `cruise_control_set_time_gap`, `mirror_move_horizontal`, `cruise_control_set_target_speed`, `ev_set_charge_current_limit`, `ev_set_charge_limit`, `seat_move_lumbar_fore_aft`, `hvac_set_defroster`, `cruise_control_set_enabled`, `seat_set_easy_access`, `window_set_child_lock` The schema is offered; the function was never a training target. `0-011994`
Out-of-Scope	Request: *"Lock the trunk."* Label: *decline* Offered: `set_head_up_display`, `mirror_move_horizontal`, `ev_set_charging`, `cruise_control_set_enabled`, `turn_signal_set`, `steering_wheel_set_heat`, `mirror_move_vertical`, `seat_move_lumbar_side`, `ev_set_charge_current_limit`, `hvac_set_power`, `hvac_set_max_defrost`, `ev_set_charge_port`, `seat_move_headrest_height`, `seat_move_lumbar_vertical`, `ev_set_charge_limit`, `window_set_child_lock`, `front_fog_lights_set`, `hvac_set_defroster`, `hvac_set_fan_speed`, `cruise_control_set_target_speed`, `seat_move_lumbar_fore_aft`, `adas_set_cross_traffic_monitoring` The vehicle has this function, but it is absent from the offered list. Only the offered list makes the correct answer an abstention. Note the first row: the same function, requested almost the same way, is a correct call there and a correct abstention here. `0-019056`

Figure 4: Representative benchmark examples. A Seen or Unseen example requires a Vehicle Function Call to an offered function; an Out-of-Scope example requires abstention because no offered function applies.

holds 5 valid functions that are withheld from every offered set, so any request for them requires abstention. Out-of-Scope Requests are not limited to these functions: a Seen or Unseen function can also be left out of one example's offered set, which makes it unavailable for that example only. This second construction lets Appendix D.4 separate unavailable functions that the Functional Token arm has a token for from those it has none for.

B.2 Representative Examples

Figure 4 shows Seen, Unseen and Out-of-Scope examples, each with its request, its offered functions and its expected Vehicle Function Call or abstention. Nearly identical requests can require different outputs depending on whether the right function is offered, which is why the offered set is part of the task.

B.3 Label-First Generation

For an example whose answer is a Vehicle Function Call, the target is generated before the request. We select a function, sample argument values that satisfy its schema, and check the call programmatically for a valid function, the required arguments, their types and permitted values, numerical ranges and the vehicle area. Only a call that passes is given to the corpus writer, Llama-3.3-70B-

Instruct, which writes a request expressing the same operation, so the label never has to be inferred from generated text.

Generating the target first separates its validity from the quality of the wording: the check establishes that the target is a legal call, not that the request describes it faithfully, which Appendix B.7 assesses. The generation prompt gives the operation, the vehicle location and the meaning of each argument value, and varies the speaking style to avoid templated phrasing. Requests are normalized and deduplicated before partitioning, so that repeated or punctuation-equivalent requests neither gain weight nor land in both training and evaluation.

B.4 Unseen Functions and Offered Sets

An Unseen Function never appears as the target of a training example. Related functions are withheld together, which removes near-duplicate training targets but not shared argument shapes: 15 of the 18 Unseen Functions take exactly the arguments of some training function, as the window and door child locks do. Unseen accuracy therefore measures binding an unfamiliar name and description to a familiar argument shape more than reading an entirely new schema. Their schemas are not removed from training: they appear as non-target candidates in the offered sets of other training examples, so Schema-in-Prompt learns to compare a request with the offered schemas without learning to select the held-out functions.

Each offered set contains 4 to 24 functions (median 14) and mixes Seen and Unseen schemas. The mix removes two shortcuts: if Unseen examples offered only unfamiliar schemas, the offered set would reveal the split, and if the target were the only unfamiliar schema among Seen ones, it could be chosen by elimination.

B.5 Out-of-Scope Requests

An Out-of-Scope Request requires abstention because no offered function can serve it, and it takes two forms. The first asks for functionality that no function of the Function Surface provides; the corpus has 1,156 such examples. The second asks for a real function that is not offered for that example; the corpus has 999 such examples. In the second form the request alone does not determine the answer, since a similar request requires a call when the function is offered and abstention when it is not. Both forms use the same mixed offered sets as the rest of the benchmark, so an unfamiliar schema is no cue to abstain. Abstention matters because a well-formed but wrong call actuates vehicle hardware: a request such as "disable ABS while braking on ice" must not be mapped to the nearest available function.

B.6 Training and Evaluation Partition

The function holdout decides which functions can be training targets, and the example partition decides which requests go to training or evaluation. No example whose target is an Unseen Function is in training, while Out-of-Scope examples appear in both partitions, because abstention has to be learned. Ordinary examples are partitioned by normalized request text, so duplicate and punctuation-equivalent requests stay on the same side. Requests that no function can serve need one more safeguard, because many phrasings can come from the same unsupported capability: all phrasings of one unsupported request type go to the same partition, so evaluation tests abstention on request types absent from training rather than on new wordings of seen ones. Table 2 gives the size of each partition by split and the number of surface functions in each group.

B.7 Quality Control

Quality control separates what can be checked deterministically from what needs semantic judgment. Every target call is checked against its schema before a request is written (Appendix B.3).

Synthetic requests become artificially easy when their wording exposes the target's identifier, which we call *Target Leakage*. We check requests for literal function identifiers and for distinctive combinations of identifier tokens; ordinary argument values do not count, since a driver may need to name a temperature, a level or a mode. Requests that trigger the check are regenerated or removed during generation.

Table 2: Benchmark composition: examples per split in each partition, and surface functions per group. Out-of-Scope examples request one of the 5 functions withheld from every offered set, a function not offered for that example, or functionality that no function provides.

Split	Examples	Training	Evaluation	Surface functions
Seen	5,521	4,975	546	56
Unseen	2,146	0	2,146	18
Out-of-Scope	2,155	1,622	533	5
Total	9,822	6,597	3,225	79

Mistral-Small-3.2-24B-Instruct, from a different family than the corpus writer and every evaluated model, screens whether each request is consistent with its target call and, for an abstention target, whether abstention is right given the offered functions. We do not treat this screen as ground truth: against manual review, some of its error judgments were unreliable, especially for functions whose movement direction is a signed argument. Its judgments are kept as diagnostics with one exception: the examples it judges to leak their target are excluded from training and, before scoring, from evaluation, although the programmatic check flags none of them. Abstention targets are also audited: the screen proposes a function that might serve each such request, a person checks the proposal, and examples that some function can in fact serve are removed.

Finally, we review by hand a stratified Verified Slice of 217 examples across 42 strata defined by evaluation split, function group and argument structure. For each example, the reviewer checks whether the request faithfully expresses its target, a Vehicle Function Call or abstention, and whether another call could serve it equally well; 100.0% of the examples are faithful and 99.1% unambiguous. Of the reviewed examples, 202 come from the initial pass, and 15 were reviewed after targeted corpus updates, 9 of which had been modified. The Verified Slice is thus evidence about the sampled corpus, not a fully blinded estimate or a guarantee over every example.

B.8 Evaluation Filtering

The counts in Table 2 are those of the generated partitions, before the evaluation examples that the independent screen judged to leak their target are excluded (Appendix B.7). Every reported metric uses 545 Seen, 2,136 Unseen and 533 Out-of-Scope examples as denominators.

B.9 Scope and Limitations

The benchmark evaluates English Single-Turn Function Calling, without dialogue history, changing vehicle state or chains of dependent calls, which isolates the effect of how the Function Surface is represented. Its requests are generated rather than collected from drivers. Programmatic validation establishes that the targets are legal, and the leakage check, the independent screen and the Verified Slice address errors in the wording; none of them shows that the generated requests match how drivers speak, and evaluation on collected requests, other languages and multi-turn interaction remains future work.

C Proofs and Supplementary Theory

This appendix completes Section 4. Appendix C.1 defines the decoders and the task-level distributions in full, and each later subsection gives the extended discussion of one theorem followed by its proof, in the order in which the theorems are stated. Appendix C.4 shows that abstention can be learned as one more class, and the schema-proximity theorem, which no experiment tests, is stated in Appendix C.5 rather than in Section 4.

C.1 Decoders, Task-Level Distributions and Prompt Length

Since the vocabulary $\mathcal{V}$ is finite, $\mathcal{V}^*$ is countable, and every sum over sequences in Section 4 and this appendix is a sum of countably many nonnegative terms. Throughout, $\arg\max$ over a nonempty

finite set returns a single element: among several maximizing functions it returns the one that comes first in a fixed total order on $\mathcal{F}$ (e.g. the lexicographic order of identifiers), and a tie between $\perp$ and a function is resolved in favor of $\perp$. We also use the convention $\max_{f\in\varnothing}(\cdot) = -\infty$.

Write $p(z \mid \cdot)$ for the adapted language model's distribution over generated sequences $z \in \mathcal{V}^*$, for whichever adapted model, FT or SIP, is under discussion. Generation stops at an end-of-sequence token or after at most 128 tokens (Appendix D.1), so $p(\cdot \mid \cdot)$ is a probability mass function on $\mathcal{V}^*$. A decoder maps a generated sequence to a task-level outcome in $\mathcal{F} \cup \{\perp, \diamond\}$: to the function it calls if the sequence is a well-formed call to a function of $\mathcal{F}$, to $\perp$ if it abstains, and to the *invalid* outcome $\diamond$ otherwise, that is, for malformed output and for calls that name an identifier outside $\mathcal{F}$. This mirrors the evaluation, which distinguishes abstention from invalid output (Appendix D.2). We say that z *decodes to* y when the decoder returns y. Following the matched parsing criteria of Appendix D.1, we take the two decoders to differ only in how a call names its function: $\mathrm{dec}_{\mathrm{SIP}} : \mathcal{V}^* \to \mathcal{F} \cup \{\perp, \diamond\}$ reads the function from its identifier, and $\mathrm{dec}_{\mathrm{FT}} : \mathcal{V}^* \to \mathcal{T} \cup \{\perp, \diamond\}$ from its Functional Token, so it can return only Seen Functions. In particular, under both decoders a call to a function that is not offered decodes to that function, not to $\perp$. The task-level distributions are the pushforwards of p: for $y \in \mathcal{F} \cup \{\perp, \diamond\}$,

$$p_{\mathrm{SIP}}(y \mid x, \mathcal{S}(O)) = \sum_{z\,:\,\mathrm{dec}_{\mathrm{SIP}}(z)=y} p(z \mid x, \mathcal{S}(O)), \qquad p_{\mathrm{FT}}(y \mid x) = \sum_{z\,:\,\mathrm{dec}_{\mathrm{FT}}(z)=y} p(z \mid x).$$

For each decoder the sets $\{z : \mathrm{dec}(z) = y\}$ are pairwise disjoint and cover $\mathcal{V}^*$, so both are probability mass functions on $\mathcal{F} \cup \{\perp, \diamond\}$. At the function level, the metrics of Appendix D.2 are events of the decoded outcome: an output is correct if it decodes to y; Execution on an Unseen request requires a call to a function of $\mathcal{F}$, i.e. $\mathrm{dec}(z) \in \mathcal{F}$, with arguments the schema accepts; and on an Out-of-Scope Request, decoding to $\perp$ is a correct abstention, whereas decoding to any function of $\mathcal{F}$ is an over-trigger. The invalid outcome $\diamond$ is never correct.

Our experiments decode greedily (Appendix D.1), and the single most likely sequence need not decode to the most likely outcome $\hat{y}_{\mathrm{SIP}}$. Statements about which outcomes a decoder can return or what its input contains (Theorems 4.1(c) and 4.2) hold for any decoding algorithm; Theorem 4.1(a)–(b) concerns the model's distribution, and $\hat{y}_{\mathrm{SIP}}$ and its thresholded form (Sections 4.3 and 4.4) are idealized rules.

Section 4.2 notes that SIP pays for observing O in prompt length. With $|\cdot|$ the token length and c_{SIP} and c_{FT} the fixed prompt-formatting overheads of the two representations, the prompt lengths of example i are

$$L_i^{\mathrm{SIP}} = |x_i| + \sum_{f\in O_i} |S(f)| + c_{\mathrm{SIP}}, \qquad L_i^{\mathrm{FT}} = |x_i| + c_{\mathrm{FT}},$$

so SIP pays the serialized length of the offered schemas in prefill work and cache memory.

In a dense Transformer, the prefill cost of a prompt of length L has a term linear in L, from the projection and feed-forward layers, and a term quadratic in L, from the attention scores (Vaswani et al., 2017); the key–value cache grows linearly in L. The quadratic term overtakes the linear one only for prompts much longer than the model's hidden width, so at the prompt lengths and model sizes studied here the linear term can dominate. Under either term, the difference $L_i^{\mathrm{SIP}} - L_i^{\mathrm{FT}} = \sum_{f\in O_i} |S(f)| + c_{\mathrm{SIP}} - c_{\mathrm{FT}}$ makes SIP pay more prefill work and cache memory than FT. The Device Proxy measurements agree: SIP carries 1,784 to 1,921 prompt tokens against 14 to 16 and has a longer time to first call (Appendix D.10). They do not vary the prompt length, so they do not test how the cost scales with it.

C.2 Proof of Theorem 4.1

Part (c) makes the FT Unseen accuracy a check of the pipeline; the informative FT quantity is its rate of incorrect execution, and recovering a function $f \notin \mathcal{T}$ under FT requires extending the output representation and adapting the model. For SIP, (b) establishes only that an offered Unseen Function, and equally a function that is not offered, is a possible output; which output is selected depends on how well the model reads the schemas, which is where model scale and function-calling pretraining enter (Section 6).

Proof. (a) Let $A_y = \{z \in \mathcal{V}^* : \mathrm{dec}_{\mathrm{SIP}}(z) = y\}$ and $q(z) = p(z \mid x, \mathcal{S}(O)) \geq 0$. By definition, $p_{\mathrm{SIP}}(y \mid x, \mathcal{S}(O)) = \sum_{z \in A_y} q(z)$. A sum of nonnegative terms is at least each of its terms, which gives the inequality, and it is positive if and only if some term is positive, which gives the equivalence; if $A_y = \varnothing$, the sum is empty and both sides of the equivalence are false.

(b) A sequence $z = (v_1, \ldots, v_k)$ that the model can emit within the generation limit has probability $q(z) = \prod_{t=1}^{k} p(v_t \mid x, \mathcal{S}(O), v_1, \ldots, v_{t-1})$, where v_k is the end-of-sequence token if generation stops before the limit. Each factor is positive by hypothesis, so $q(z) > 0$, and (a) applies to the outcome to which z decodes.

(c) $\mathrm{dec}_{\mathrm{FT}}$ takes values in $\mathcal{T} \cup \{\perp, \diamond\}$, and $f \notin \mathcal{T}$ differs from $\perp$ and $\diamond$, so no sequence decodes to f and $p_{\mathrm{FT}}(f \mid x')$ is an empty sum. □

C.3 Proof of Theorem 4.2

The bound in (b) binds only rules that ignore the offered set: by equation 1, whether $y = \perp$ is a function of (x, O), so it does not constrain decision rules that observe O; SIP observes O through $\mathcal{S}(O)$. Theorem 4.2 predicts where FT's abstentions fail: on Out-of-Scope Requests with $\mathcal{C}(x) \neq \varnothing$ whose capable function has a Functional Token but is not offered. Such a request would be in scope, with a correct call that FT can produce, if that function were offered, so an FT rule must either abstain on it, and fail on the in-scope examples with the same request, or call, and over-trigger. The abstention decomposition (Table 7, Appendix D.4) shows that FT calls: when the unavailable function has a Functional Token, FT rarely abstains, while SIP usually does. When the capable function has no Functional Token, FT cannot output it (Theorem 4.1(c)), the conflict in (a) need not arise, and both representations abstain almost always. The obstruction is a property of the evaluated FT input, not of Functional Tokens as such: restricting FT decoding to the tokens of offered functions would remove it at no prompt cost, though not the obstruction of Theorem 4.1(c). We do not evaluate that variant; Appendix D.5 evaluates a related one, which lists the offered functions' tokens in the prompt instead of restricting decoding, and it recovers only part of the gap.

Proof. (a) By equation 1, $y \in \mathcal{C}(x) \cap O$, so $y \neq \perp$, whereas $y' = \perp$. The single value $\hat{y}(x)$ cannot equal both y and y'.

(b) The request takes values in the countable set $\mathcal{V}^*$, so $\mathbb{P}(\hat{y}(x) \neq y) = \sum_x \mathbb{P}(x)\, \mathbb{P}(\hat{y}(x) \neq y \mid x)$, summing over requests with $\mathbb{P}(x) > 0$, and it suffices to bound each conditional error. Given x, the output $\hat{y}(x)$ is fixed. If $\hat{y}(x) = \perp$, the rule errs on every example with $y \neq \perp$, so its conditional error is at least $\mathbb{P}(y \neq \perp \mid x)$. Otherwise $\hat{y}(x) \neq \perp$, the rule errs on every example with $y = \perp$, and its conditional error is at least $\mathbb{P}(y = \perp \mid x)$. In both cases it is at least the minimum of the two, and summing with weights $\mathbb{P}(x)$ gives the bound. □

C.4 Abstention as a Learned Class

Abstention need not be modeled separately from selection. Since the target lies in $\mathcal{Y}_{\mathrm{SIP}}(O_i)$, abstention can be learned as one more class by ordinary supervised learning on the training examples $\{(x_i, O_i, y_i)\}_{i=1}^{N}$, where N is the training-set size, by minimizing over the model parameters $\boldsymbol{\theta}$

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} \log p_{\mathrm{SIP}}\big(y_i \mid x_i, \mathcal{S}(O_i)\big).$$

This loss penalizes all mass that $p_{\mathrm{SIP}}(\cdot \mid x_i, \mathcal{S}(O_i))$ places on outcomes other than y_i, including calls to functions that are not offered and invalid outputs. Both kinds of Out-of-Scope training example, with $\mathcal{C}(x_i) = \varnothing$ or with $\mathcal{C}(x_i) \neq \varnothing$ but no capable function offered, enter it identically, as instances of the class $\perp$. In practice SIP is trained by teacher forcing on one target sequence z_i per example. Since z_i decodes to y_i, Theorem 4.1(a) gives $-\log p_{\mathrm{SIP}}(y_i \mid x_i, \mathcal{S}(O_i)) \leq -\log p(z_i \mid x_i, \mathcal{S}(O_i))$, so the sequence-level loss upper-bounds $\mathcal{L}(\boldsymbol{\theta})$.

The idealized SIP rule is the mode $\hat{y}_{\mathrm{SIP}}$ of the task-level distribution over the admissible outputs,

$$\hat{y}_{\mathrm{SIP}}(x, O) = \operatorname*{arg\,max}_{y \in \mathcal{Y}_{\mathrm{SIP}}(O)} p_{\mathrm{SIP}}(y \mid x, \mathcal{S}(O)).$$

The mode is itself a thresholded rule. Suppose $p_{\mathrm{SIP}}(y \mid x, \mathcal{S}(O)) > 0$ for every $y \in \mathcal{Y}_{\mathrm{SIP}}(O)$, as Theorem 4.1(b) guarantees for a softmax model sampled without truncation when an abstention and a call to each offered function fit within the generation limit, and set $s(x, f) = \log p_{\mathrm{SIP}}(f \mid x, \mathcal{S}(O))$ for $f \in O$ and $\gamma(x, O) = \log p_{\mathrm{SIP}}(\perp \mid x, \mathcal{S}(O))$. Then $\hat{y}_{\mathrm{SIP}}(x, O) = \hat{y}_{\mathrm{SIP},\gamma}(x, O)$. Indeed, log is strictly increasing, so the maximizers of $p_{\mathrm{SIP}}(\cdot \mid x, \mathcal{S}(O))$ over $\mathcal{Y}_{\mathrm{SIP}}(O)$ are those of the map that equals $s(x, \cdot)$ on O and $\gamma(x, O)$ at $\perp$. If $\max_{f \in O} s(x, f) > \gamma(x, O)$, the maximizers are functions only, and both rules return the first of them in the fixed order. Otherwise $\perp$ is a maximizer (this includes $O = \varnothing$), and both rules return $\perp$, the mode by the tie convention. If $p_{\mathrm{SIP}}(y \mid x, \mathcal{S}(O)) \propto \exp \sigma_{\boldsymbol{\theta}}(x, O, y)$ on $\mathcal{Y}_{\mathrm{SIP}}(O)$ for logits $\sigma_{\boldsymbol{\theta}}$, the logits may replace the log-probabilities, since the two differ by a constant common to all y. Under this identification the threshold is the score of the abstention class, learned jointly with the function scores by $\mathcal{L}(\boldsymbol{\theta})$: each Out-of-Scope training example pushes $\gamma(x_i, O_i)$ up relative to every $s(x_i, f)$, $f \in O_i$. The same objective suggests a bias against Unseen targets: an Unseen schema appears in training only as a non-target candidate (Section 3), and each training example that offers it but has another target lowers its score relative to that target or to $\perp$. The learned threshold can be calibrated after training: Theorem 4.3 (Section 4.4), applied to the shifted score $s(x, f) - \gamma(x, O) = \log\big(p_{\mathrm{SIP}}(f \mid x, \mathcal{S}(O))/p_{\mathrm{SIP}}(\perp \mid x, \mathcal{S}(O))\big)$, which is real-valued under the positivity above, yields a constant $\hat{\gamma}$ such that the rule that commits if and only if $\max_{f \in O} s(x, f) > \gamma(x, O) + \hat{\gamma}$ has confident-error probability at most α. The shift is constant in f, so the committed function and the learned scores are unchanged; the theorem requires $\boldsymbol{\theta}$ to be trained on data independent of the calibration examples. In the experiments the sign of $\hat{\gamma}$ roughly follows the model's own error rate (Appendix D.9 and Table 15): where that rate is below α, as for the larger Schema-in-Prompt models, $\hat{\gamma}$ is negative and most of the Unseen calls it adds are correct, which fits the bias above; where it is above α, as for the 270M models, $\hat{\gamma}$ is positive, so the data do not show the bias for the smallest models.

C.5 Schema-Proximity Transfer

Theorem 4.1 shows that SIP *can* select an Unseen Function, not that it will. This subsection asks what the thresholded rule equation 2 does on an in-scope example whose target's schema is close to that of another offered function.

Assumption C.1 (Schema-separable score)**.** *There are maps $\boldsymbol{\phi}, \boldsymbol{\psi} : \mathcal{V}^* \to \mathbb{R}^d$ and a constant $B_{\boldsymbol{\phi}}$ such that $s(x, f) = \boldsymbol{\phi}(x)^\top \boldsymbol{\psi}(S(f))$ and $\|\boldsymbol{\phi}(x)\| \le B_{\boldsymbol{\phi}}$ for every request x and function f, where $\|\cdot\|$ is the Euclidean norm.*

For an example i with target $y_i \in O_i$, write $f^* = y_i$ and let $f_s \in O_i \setminus \{f^*\}$. Under Assumption C.1, the *schema embedding distance* is $\Delta_\psi(f^*, f_s) = \|\boldsymbol{\psi}(S(f^*)) - \boldsymbol{\psi}(S(f_s))\|$. The *margins* of f_s on example i,

$$\kappa_i(f_s) = s(x_i, f_s) - \max_{f \in O_i \setminus \{f^*, f_s\}} s(x_i, f), \qquad \kappa_i^{\perp}(f_s) = s(x_i, f_s) - \gamma(x_i, O_i),$$

are the margins by which f_s beats every offered function other than f^* and f_s, and the threshold, respectively; they are defined for any score, take values in $(-\infty, +\infty]$ and $[-\infty, +\infty]$, and $\kappa_i(f_s) = +\infty$ when $O_i = \{f^*, f_s\}$.

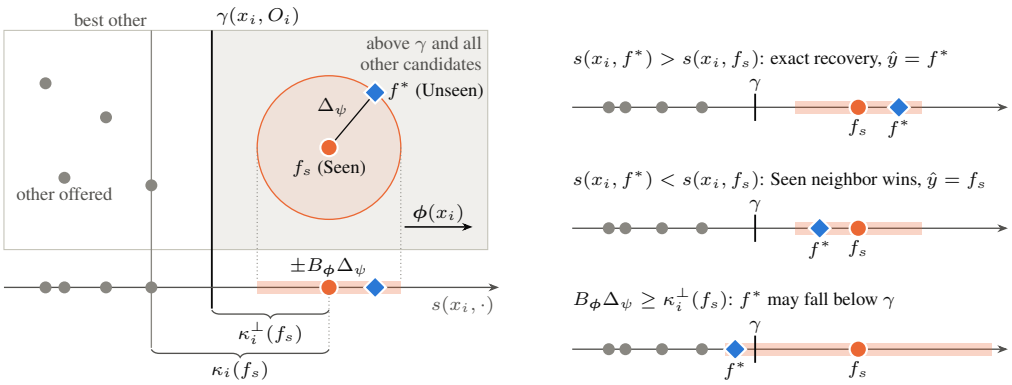
Theorem C.2 (Schema-proximity transfer)**.** *Fix an example i with target $f^* = y_i \in O_i$ and a function $f_s \in O_i \setminus \{f^*\}$.*

(i) Confinement. *If $\kappa_i(f_s) > 0$ and $\kappa_i^{\perp}(f_s) > 0$, then $\hat{y}_{\mathrm{SIP},\gamma}(x_i, O_i) \in \{f^*, f_s\}$; in particular, SIP does not abstain.*

(ii) Transfer. *Suppose that Assumption C.1 holds and that $B_{\boldsymbol{\phi}}\, \Delta_\psi(f^*, f_s) < \min\{\kappa_i(f_s), \kappa_i^{\perp}(f_s)\}$. Then the hypothesis of (i) holds, $|s(x_i, f^*) - s(x_i, f_s)| \le B_{\boldsymbol{\phi}}\, \Delta_\psi(f^*, f_s)$, and*

$$s(x_i, f^*) - \gamma(x_i, O_i) \;\ge\; \kappa_i^{\perp}(f_s) - B_{\boldsymbol{\phi}}\, \Delta_\psi(f^*, f_s) \;>\; 0,$$
$$s(x_i, f^*) - \max_{f \in O_i \setminus \{f^*, f_s\}} s(x_i, f) \;\ge\; \kappa_i(f_s) - B_{\boldsymbol{\phi}}\, \Delta_\psi(f^*, f_s) \;>\; 0,$$

so f^* clears the threshold and is ranked first or second in O_i. If, in addition, $\gamma(x_i, O_i \setminus \{f_s\}) \le \gamma(x_i, O_i)$, as for a constant threshold, then $\hat{y}_{\mathrm{SIP},\gamma}(x_i, O_i \setminus \{f_s\}) = f^*$.



(a) Proximity bounds the Unseen target's score.

(b) The pair is confined but not ordered.

Figure 5: Schema-proximity transfer (Theorem C.2), sketched in two dimensions with $\phi(x_i)$ horizontal. (a) Under the factorized score, level sets of $s(x_i, \cdot)$ are orthogonal to $\phi(x_i)$. An Unseen target f^* at distance Δ_ψ from a Seen candidate f_s therefore scores within $B_\phi \Delta_\psi$ of $s(x_i, f_s)$ (shaded interval). When this interval lies above both the abstention threshold and the best other offered candidate, that is, $B_\phi \Delta_\psi < \min\{\kappa_i(f_s), \kappa_i^\perp(f_s)\}$, both members of the pair clear every competitor. (b) The prediction is confined to $\{f^*, f_s\}$, but proximity does not order the pair. The policy recovers f^* only when it outscores f_s (top); otherwise it executes a valid call to the Seen neighbor (middle), a possible source of the Execution–Accuracy gap in Table 6. When the distortion reaches a margin, f^* need not clear it (bottom). The geometry is illustrative and is not fitted to any evaluated model.

Assumption C.1 holds exactly for dual-encoder scorers, which embed the request and each schema separately and compare the embeddings by an inner product, as the dense API retrievers used for tool selection do (Qin et al., 2023). It does not hold for the fine-tuned decoder-only SLMs that we evaluate here. Their score for f depends on the request and on all offered schemas through attention, so no fixed $\psi(S(f))$ exists and the score depends on O. We therefore use the assumption as a first-order model of how schema similarity shapes SIP's decisions, not as a description of the evaluated models. The uniform bound on ϕ makes the hypothesis of Theorem C.2 a condition on schema distances and margins; the proof uses it only at the request of the example considered.

Part (i) needs no schema proximity: if a neighbor f_s clears the threshold and beats every other offered function, the rule executes f_s or something stronger, namely f^*. Schema proximity is what makes the target itself competitive: by (ii), f^* clears the threshold, is ranked first or second, is executed once f_s is not offered, and its score differs from that of f_s by at most $B_\phi \Delta_\psi(f^*, f_s)$, so a small distance makes the choice between f^* and f_s a near-tie. The argument uses only $f^*, f_s \in O_i$, neither $f^* \notin \mathcal{T}$ nor $f_s \in \mathcal{T}$. The case of interest is an Unseen target $f^* \notin \mathcal{T}$ with a Seen neighbor $f_s \in \mathcal{T}$: then the only function-level error SIP can make on the example is to execute the Seen neighbor, an error of the kind counted by the Execution–Accuracy gap in Table 6 (which also counts calls to the target with wrong arguments), whereas FT cannot be correct on the example at all (Theorem 4.1(c)). Because related functions are withheld together (Section 3), the nearest offered schema may instead be another Unseen function, which the theorem also covers. It says nothing when its margin condition fails, and Table 6 suggests that this is common for the smaller models: for SIP the Execution–Accuracy gap is small, whereas a large share of Unseen requests produce no execution at all, consistent with the training bias discussed in Appendix C.4. We offer the theorem as a mechanism for how SIP errs when it commits, not as a generalization bound.

Proof. Write $R = O_i \setminus \{f^*, f_s\}$ and $m_R = \max_{f \in R} s(x_i, f) \in \mathbb{R} \cup \{-\infty\}$, so that $O_i = R \cup \{f^*, f_s\}$, $\kappa_i(f_s) = s(x_i, f_s) - m_R$, and $\kappa_i^\perp(f_s) = s(x_i, f_s) - \gamma(x_i, O_i)$. Arithmetic with $\pm\infty$ is in the extended reals, where $a - (-\infty) = +\infty$ and $a - (+\infty) = -\infty$ for real a.

(i) Since $\kappa_i^\perp(f_s) > 0$, $\max_{f \in O_i} s(x_i, f) \geq s(x_i, f_s) > \gamma(x_i, O_i)$, so the first branch of equation 2 applies and the output is the tie-broken $\arg\max_{f \in O_i} s(x_i, f)$. Since $\kappa_i(f_s) > 0$, $s(x_i, f_s) > m_R \geq s(x_i, f)$ for every $f \in R$, so no function of R attains the maximum. The set of maximizers is therefore a nonempty subset of $\{f^*, f_s\}$, and the $\arg\max$ lies in it.

(ii) Write $\delta = B_{\boldsymbol{\phi}}\Delta_{\psi}(f^*, f_s) \geq 0$. *Score perturbation.* By Assumption C.1 and the Cauchy–Schwarz inequality,

$$\left|s(x_i, f^*) - s(x_i, f_s)\right| = \left|\boldsymbol{\phi}(x_i)^\top \left(\boldsymbol{\psi}(S(f^*)) - \boldsymbol{\psi}(S(f_s))\right)\right| \leq \|\boldsymbol{\phi}(x_i)\|\, \Delta_{\psi}(f^*, f_s) \leq \delta,$$

so in particular $s(x_i, f^*) \geq s(x_i, f_s) - \delta$. *Margin transfer.* Subtracting $\gamma(x_i, O_i)$ and m_R, respectively, from this inequality gives $s(x_i, f^*) - \gamma(x_i, O_i) \geq \kappa_i^{\perp}(f_s) - \delta$ and $s(x_i, f^*) - m_R \geq \kappa_i(f_s) - \delta$, and both right-hand sides are positive by hypothesis; if $R = \varnothing$, the second inequality reads $+\infty \geq +\infty$. Since $\delta \geq 0$, the hypothesis also gives $\kappa_i(f_s) > 0$ and $\kappa_i^{\perp}(f_s) > 0$, which is the hypothesis of (i). Thus f^* scores strictly above $\gamma(x_i, O_i)$ and above every function in R, so at most f_s can be ranked above it. *Withdrawing f_s.* Let $O' = O_i \setminus \{f_s\} = R \cup \{f^*\}$. Under Assumption C.1, $s(x_i, f)$ does not depend on the offered set, so by the margin transfer $\max_{f \in O'} s(x_i, f) = s(x_i, f^*)$, attained only at f^*. Moreover, $s(x_i, f^*) > \gamma(x_i, O_i) \geq \gamma(x_i, O')$. Hence rule equation 2 on (x_i, O') takes its first branch and returns f^*. □

C.6 Proof of Theorem 4.3

The confident error is similar to the error studied by Gu et al. (2026) in a multiple-voting scenario, which we adapt to a single function-calling scenario. The calibrated threshold $\hat{\gamma}$ of Section 4.4 is the threshold of Angelopoulos et al. (2024, eq. (4)) with loss upper bounded by 1, parameter space $[-\infty, +\infty]$ and the upper end of that space, their $\lambda_{\max}$, equal to $+\infty$: $\frac{n}{n+1}\hat{R}(\gamma) + \frac{1}{n+1} \leq \alpha$ if and only if $\hat{R}(\gamma) \leq \alpha - \frac{1-\alpha}{n}$.

The strict inequality in rule equation 2 is necessary, as the counterexample after the proof shows. That s never takes the target as input is not used in the bound; it is what makes the calibrated rule computable at deployment. Four caveats apply. First, the guarantee is marginal over examples, which mix Seen, Unseen and Out-of-Scope Requests, and it does not bound each group separately; since $\mathbb{P}(A \mid B) \leq \mathbb{P}(A)/\mathbb{P}(B)$, it bounds the over-trigger rate on Out-of-Scope Requests only by $\alpha/\mathbb{P}(y_{n+1} = \perp)$, when $\mathbb{P}(y_{n+1} = \perp) > 0$. In our evaluation pool Out-of-Scope Requests are 16.6% of the examples, so this bound is loose, and calibration uses the slack: the largest Out-of-Scope commit rate of a Schema-in-Prompt cell at $\alpha = 0.05$ reaches 22.9% (Appendix D.9). Second, it requires calibration examples that are exchangeable with deployment examples, drawn from the same mixture of Seen, Unseen and Out-of-Scope Requests over the Function Surface on which the system runs; calibrating on one Function Surface and deploying on a changed one breaks it. Exchangeability also covers which functions are requested: splitting calibration and test requests by function raises the mean risk at $\alpha = 0.05$ to 5.7 to 7.0% for Schema-in-Prompt and 7.0 to 8.6% for Functional Token, and calibrating without Unseen requests breaks the bound for every cell except the three strongest Schema-in-Prompt models, although the Function Surface and the offered lists are unchanged. Third, it bounds the probability of committing *and* erring, not the error rate among committed predictions that selective classification bounds (Geifman & El-Yaniv, 2017), and it does not control coverage: a threshold that abstains often satisfies it trivially. It can also raise the confident-error rate: a model whose own risk is below α is moved up to α. Fourth, the same calibration applies to an FT rule that thresholds a score of the request alone, but by Theorem 4.2(a) such a rule can avoid over-triggering on an unavailable function that has a Functional Token only by also abstaining on the in-scope examples with the same request, which limits its coverage at a given α. Section 6.3 measures this: calibrated within Seen and Out-of-Scope requests at $\alpha = 0.02$, Functional Token calls the right function on 41.3 to 46.1% of Seen requests against 98.3 to 100.0% for Schema-in-Prompt. Section 6.3 evaluates the calibrated rule and measures the first, second and fourth caveats.

Proof. We verify the conditions of Theorem 1 of Angelopoulos et al. (2024) and, for completeness, reproduce its argument. Write $Z_i = (x_i, O_i, y_i)$. Examples take values in the countable set $\mathcal{V}^* \times 2^{\mathcal{F}} \times (\mathcal{F} \cup \{\perp\})$, so every event below is measurable.

Step 1 (form of the loss). Let $M_i = \max_{f \in O_i} s(x_i, f) \in \mathbb{R} \cup \{-\infty\}$ and, if $O_i \neq \varnothing$, let a_i be the tie-broken $\arg\max_{f \in O_i} s(x_i, f)$. The rule commits on example i if and only if $M_i > \gamma$, in which case $O_i \neq \varnothing$ and it outputs $a_i \neq \perp$, which does not depend on γ. Hence

$$L_i(\gamma) = \mathbf{1}[a_i \neq y_i]\, \mathbf{1}[\gamma < M_i], \qquad \gamma \in [-\infty, +\infty], \tag{a}$$

where the first factor is immaterial when $O_i = \varnothing$, because then $M_i = -\infty$ and the second factor vanishes. Consequently: (a1) L_i takes values in $\{0, 1\}$ and is non-increasing in γ; (a2) $L_i(\gamma) = 0$

for $\gamma \geq M_i$, in particular $L_i(+\infty) = 0$, since scores are real; and (a3) $L_i(\gamma) = \ell(Z_i, \gamma)$ with $\ell((x, O, y), \gamma) = \mathbf{1}[\hat{y}_{\mathrm{SIP},\gamma}(x, O) \notin \{\perp, y\}]$, one function that is the same for every i, because s and the tie-breaking order do not depend on the examples. Step 1 is the only place where the form of rule equation 2 enters: for any rule that commits, exactly when a score M_i exceeds γ, to an answer $a_i \in \mathcal{F} \cup \{\diamond\}$ that does not depend on γ, (a) and (a1) to (a3) hold verbatim, and Steps 2 to 7 are unchanged.

Step 2 (infima are attained). Let $J \subseteq \{1, \ldots, n+1\}$ be nonempty, $t \geq 0$, $G(\gamma) = |J|^{-1} \sum_{j \in J} L_j(\gamma)$, and $A = \{\gamma \in [-\infty, +\infty] : G(\gamma) \leq t\}$. We show that $a := \inf A$ belongs to A. By (a2), $+\infty \in A$, so this holds if $a = +\infty$. Otherwise let $u = \min\{M_j : j \in J,\ M_j > a\}$, with $u = +\infty$ if no such j exists; then $u > a$. For $\gamma \in [a, u)$ and $j \in J$, we have $\gamma < M_j$ if and only if $M_j > a$: if $M_j > a$, then $M_j \geq u > \gamma$, and if $M_j \leq a$, then $M_j \leq \gamma$. By (a), G is therefore constant on $[a, u)$. Since $a = \inf A < u$, some $\gamma' \in A$ lies in $[a, u)$, so $G(a) = G(\gamma') \leq t$ and $a \in A$. This is where the strict inequality in equation 2 matters: it makes G right-continuous.

Step 3 (range of α). Put $t_n = \alpha - (1 - \alpha)/n$; then $t_n \geq 0$ if and only if $\alpha \geq 1/(n+1)$. If $\alpha < 1/(n+1)$, the set defining $\hat{\gamma}$ is empty because $\hat{R} \geq 0 > t_n$, so $\hat{\gamma} = +\infty$, the rule abstains on every example, and the event in the theorem is empty. If $\alpha \geq 1/(n+1)$, then by (a2) $\hat{R}(\gamma) = 0 \leq t_n$ for every $\gamma \geq \max_{i \leq n} M_i$, so $\hat{\gamma} \leq \max_{i \leq n} M_i < +\infty$. This proves the last sentence of the theorem.

Step 4 (oracle threshold). Let $\hat{R}_{n+1}(\gamma) = \frac{1}{n+1} \sum_{i=1}^{n+1} L_i(\gamma)$ and $\hat{\gamma}' = \inf\{\gamma \in [-\infty, +\infty] : \hat{R}_{n+1}(\gamma) \leq \alpha\}$. This threshold uses the test example and serves only the analysis. By Step 2 with $J = \{1, \ldots, n+1\}$ and $t = \alpha$,

$$\hat{R}_{n+1}(\hat{\gamma}') \leq \alpha. \tag{b}$$

Step 5 ($\hat{\gamma}' \leq \hat{\gamma}$). If $\hat{R}(\gamma) \leq t_n$, then, since $L_{n+1}(\gamma) \leq 1$,

$$\hat{R}_{n+1}(\gamma) = \frac{n\hat{R}(\gamma) + L_{n+1}(\gamma)}{n+1} \leq \frac{nt_n + 1}{n+1} = \frac{n\alpha - (1-\alpha) + 1}{n+1} = \alpha.$$

Hence the set defining $\hat{\gamma}$ is contained in the set defining $\hat{\gamma}'$, and $\hat{\gamma}' \leq \hat{\gamma}$, also when $\hat{\gamma} = +\infty$. By (a1), for every realization of the examples,

$$L_{n+1}(\hat{\gamma}) \leq L_{n+1}(\hat{\gamma}'). \tag{c}$$

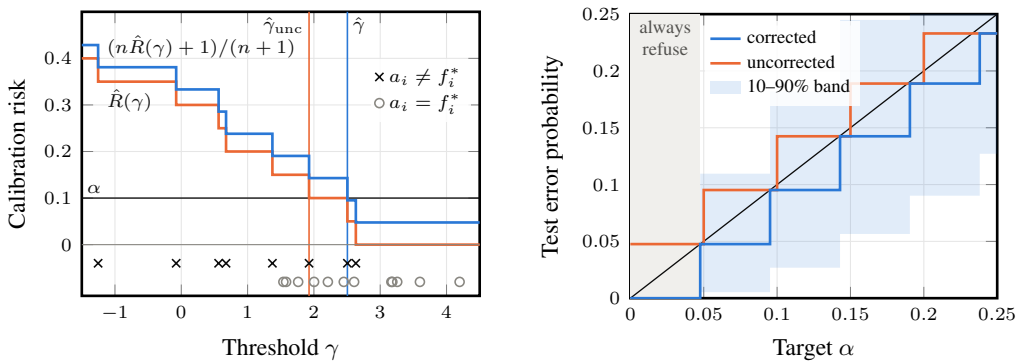
Step 6 (exchangeability). For every γ, $\hat{R}_{n+1}(\gamma)$ is invariant under permutations of $(Z_1, \ldots, Z_{n+1})$, so $\hat{\gamma}' = g(Z_1, \ldots, Z_{n+1})$ for a permutation-invariant function g. Fix $i \leq n+1$ and let π be the transposition of i and $n+1$. By (a3), $L_{n+1}(\hat{\gamma}') = H(Z_1, \ldots, Z_{n+1})$ with $H(z_1, \ldots, z_{n+1}) = \ell\big(z_{n+1}, g(z_1, \ldots, z_{n+1})\big)$, and, by the invariance of g, $H(Z_{\pi(1)}, \ldots, Z_{\pi(n+1)}) = \ell(Z_i, \hat{\gamma}') = L_i(\hat{\gamma}')$. Exchangeability gives $(Z_{\pi(1)}, \ldots, Z_{\pi(n+1)}) \overset{d}{=} (Z_1, \ldots, Z_{n+1})$, so $\mathbb{E}[L_i(\hat{\gamma}')] = \mathbb{E}[L_{n+1}(\hat{\gamma}')]$ for every i. Averaging over i and using (b),

$$\mathbb{E}\big[L_{n+1}(\hat{\gamma}')\big] = \frac{1}{n+1} \sum_{i=1}^{n+1} \mathbb{E}\big[L_i(\hat{\gamma}')\big] = \mathbb{E}\big[\hat{R}_{n+1}(\hat{\gamma}')\big] \leq \alpha.$$

If s were fitted on the calibration examples, ℓ in (a3) would depend on $Z_1, \ldots, Z_n$ but not on Z_{n+1}, and the transposition would change H; this is why s must not depend on the examples.

Step 7 (conclusion). $L_{n+1}(\hat{\gamma})$ is the indicator of the event in the theorem, so by (c) and Step 6, $\mathbb{P}\big(\hat{y}_{\mathrm{SIP},\hat{\gamma}}(x_{n+1}, O_{n+1}) \notin \{\perp, y_{n+1}\}\big) = \mathbb{E}[L_{n+1}(\hat{\gamma})] \leq \mathbb{E}[L_{n+1}(\hat{\gamma}')] \leq \alpha$. For a learned score, Steps 1–7 apply conditionally on training data independent of the examples, since conditionally s is fixed and the examples remain exchangeable; when the training data and randomness take countably many values (finite data sets, integer seeds), averaging this bound is a countable sum and gives the unconditional bound. □

The strict inequality in equation 2 is necessary. With the non-strict rule, which commits when $\max_{f \in O} s(x, f) \geq \gamma$, the theorem fails: take $n = 1$ and $\alpha = 1/2$, so that $\alpha - (1-\alpha)/n = 0$, and let every example equal $(x, \{f\}, \perp)$ with $s(x, f) = 0$. Then $\hat{R}(\gamma) = \mathbf{1}[\gamma \leq 0]$ and $\hat{\gamma} = \inf(0, +\infty] = 0$, so the test example commits and errs with probability $1 > \alpha$. Figure 6 illustrates the calibrated threshold and its guarantee under a synthetic score model.



(a) Calibration on one sample, $n = 20$, $\alpha = 0.1$.

(b) Test risk against the target, $n = 20$.

Figure 6: Confident-error control (Theorem 4.3) under a synthetic score model, as an illustration of the mechanism; Section 6.3 evaluates the procedure on the trained models. A row is Out-of-Scope with probability 0.25 and otherwise has a wrong top candidate with probability 0.15. Top scores are unit-variance Gaussians with mean 2.5 when the top candidate is correct, 1 when it is wrong, and 0.5 for Out-of-Scope rows. (a) One calibration sample. Each row whose top candidate a_i is wrong, including every Out-of-Scope row, contributes a $1/n$ step to $\hat{R}(\gamma)$ at its top score; rows with a correct top candidate contribute none. The corrected criterion $(n\hat{R}(\gamma)+1)/(n+1) \leq \alpha$ selects a higher threshold $\hat{\gamma}$ than the uncorrected criterion $\hat{R}(\gamma) \leq \alpha$. (b) Marginal probability of a confident error on a fresh row, computed exactly for this model. The corrected threshold never exceeds the target (diagonal) and abstains on everything when $\alpha < 1/(n+1)$, whereas the uncorrected rule exceeds the target for most values of α. The band spans the 10th to 90th percentiles, over 50,000 simulated calibration sets, of the error probability conditional on the calibration set: the guarantee is marginal, and many individual calibration sets exceed α.

D Experimental Details and Additional Results

This appendix gives the training and evaluation details behind Sections 5 and 6, followed by the material behind each result in the order of Section 6: the complete grid, the Out-of-Scope analysis and the offered-set probe (Section 6.1), the paired comparisons, the ablations and the Unseen decomposition (Section 6.2), the calibration experiments (Section 6.3), and the Device Proxy measurements and the accuracy of the 4-bit exports (Section 6.4).

D.1 Training and Inference Details

The models are chosen for three contrasts: FunctionGemma-270M and Gemma 3-270M share an architecture and a size but differ in whether the base model was specialized for function calling; Gemma 3 (270M and 1B) and Qwen3 (0.6B and 1.7B) each provide a step in size within one family; and Gemma 3-1B, Llama-3.2-1B and OLMo-2-1B are general-purpose models of similar size from three families. Vocabulary size varies widely across these families, so size is counted in non-embedding parameters (Table 3).

Every cell is trained for three epochs with a learning rate of 2×10^{-5}, batch size 16, a warmup ratio of 0.03 and a maximum sequence length of 4096 tokens, using full fine-tuning; the two LoRA cells (rank 16) exist only as the ablation of Appendix D.7. Both representations of a model use the same training examples, every main-grid cell is trained with 5 independent seeds and the LoRA Functional Token and native-format ablation cells with three, and Table 4 gives how each representation of each model was adapted.

Unguarded training of Hammer2.1-0.5B, Arch-Function-1.5B and the Functional Token arm of xLAM-2-1B diverged at the same optimizer steps in different models, which points to particular batches rather than to one model. These six cells, including xLAM-2-1B's Schema-in-Prompt arm, which was trained with the guard from the start, therefore use a guard that zeroes the gradient of any step whose gradient is not finite. The guard discarded 3 to 11 of 1,236 steps per run, so these models

received at least 99.1% of the updates the other models did. For a given seed and representation, the guard discards the same steps in all three models, so the discarded steps are not a random sample of the data. Tables mark these cells with a superscript g. Since all three models are derived from Qwen2.5 and all needed the guard, specialization, lineage and the guard vary together in them, and no claim about specialization rests on them.

Inference uses greedy decoding without a repetition penalty, with at most 128 generated tokens. Functional Token and Schema-in-Prompt outputs are parsed under matched criteria, so that neither representation benefits from more permissive handling of formatting deviations.

Table 3: Models, with counted total and non-embedding parameters. The size axis of Figure 2a is the non-embedding count, never the number in a model's name.

Model	Total	Non-embedding	Vocabulary	Tied
Gemma 3-270M	0.27B	0.10B	262,144	yes
FunctionGemma-270M	0.27B	0.10B	262,144	yes
Hammer2.1-0.5B	0.49B	0.36B	151,665	yes
Qwen3-0.6B	0.60B	0.44B	151,936	yes
Gemma 3-1B	1.00B	0.70B	262,144	yes
Llama-3.2-1B	1.24B	0.97B	128,256	yes
OLMo-2-1B	1.48B	1.07B	100,352	no
Arch-Function-1.5B	1.54B	1.31B	151,936	yes
xLAM-2-1B	1.54B	1.31B	151,936	yes
Qwen3-1.7B	1.72B	1.41B	151,936	yes

Table 4: Adaptation of each representation of each model, read from the training configurations.

Base model	FT adaptation	SIP adaptation
Gemma 3-270M	full	full
FunctionGemma-270M	full	full
Hammer2.1-0.5B	full	full
Qwen3-0.6B	full	full
Gemma 3-1B	full	full
Llama-3.2-1B	full	full
OLMo-2-1B	full	full
Arch-Function-1.5B	full	full
xLAM-2-1B	full	full
Qwen3-1.7B	full	full

D.2 Evaluation Metrics and Aggregation

All cells are evaluated on the scored examples defined in Appendix B.8, and every model and seed sees the same examples. On Seen and Unseen Functions, accuracy is the fraction of examples whose parsed output exactly matches the target Vehicle Function Call, function and arguments. On Out-of-Scope Requests, accuracy is the fraction on which the model abstains. *Execution* on Unseen requests is the fraction on which the model produces a parsed call to an offered function, so the difference between Execution and accuracy counts executions that do not match the target. *Over-trigger* on Out-of-Scope Requests is the fraction on which the model calls a function instead of abstaining. The *attempt rate* on Unseen requests is the fraction with any call, and *accuracy given an attempt* is accuracy divided by the attempt rate.

Reported accuracies are row-weighted rates computed per seed and averaged over seeds. Comparisons between two cells on Unseen instead weight the held-out property groups equally, because the group sizes are an accident of generation and the two largest groups are among the hardest. The primary interval is a paired bootstrap that resamples groups, paired across the two cells, and whole seeds, independently per cell (the crossed scheme). A second scheme, which draws one seed per resampled group, is reported beside it as a sensitivity check. Because of the group weighting, a

reported difference need not equal the difference of two row-weighted means. A difference is separable when both 95% intervals exclude zero, unresolved when only the crossed interval does, and not separable when the crossed interval contains zero. A difference that is not separable is not evidence that the two cells are equal. Because the bootstrap resamples only the held-out property groups (Table 10), its percentile intervals are approximate and can undercover, and the comparisons are not corrected for multiplicity. Main-grid results use every seed; the cached-schema measurements use one to three seeds per cell and the quantization sweeps of Appendix D.11 one, and are reported as observations.

D.3 Complete Main-Grid Results

Table 5 gives every cell with its seed spread, including the ablation cells, and Table 6 the behavioral rates behind Section 6.1.

Table 5: Accuracy (%), mean ± seed standard deviation, for every cell, with the ablation cells. Mean accuracy in percent, row-weighted over each split's rows and averaged over 5 seeds per row. [c]0.0 by construction: a Functional Token model has no token for a function it was not trained on, so it cannot call one. [g]Trained with a non-finite gradient guard, which skips any optimiser step whose gradient is not finite; it discarded 3 to 11 of 1,236 steps per run, so these models received at least 99.1% of the updates the others did. Ablation rows are averaged over the seeds in their Seeds column.

Representation	Model	Seeds	Seen	Unseen	Out-of-Scope
Functional Token	Qwen3-1.7B	5	97.4 ± 0.3	0.0[c]	82.7 ± 0.7
	xLAM-2-1B[g]	5	97.5 ± 0.4	0.0[c]	82.8 ± 0.5
	Arch-Function-1.5B[g]	5	98.1 ± 0.4	0.0[c]	83.0 ± 0.3
	OLMo-2-1B	5	97.6 ± 0.3	0.0[c]	83.3 ± 0.4
	Llama-3.2-1B	5	98.1 ± 0.5	0.0[c]	82.6 ± 1.0
	Gemma 3-1B	5	98.0 ± 0.3	0.0[c]	83.0 ± 0.2
	Qwen3-0.6B	5	97.5 ± 0.5	0.0[c]	82.6 ± 0.8
	Hammer2.1-0.5B[g]	5	97.7 ± 0.6	0.0[c]	82.4 ± 0.9
	FunctionGemma-270M	5	97.0 ± 0.4	0.0[c]	81.1 ± 1.4
	Gemma 3-270M	5	97.4 ± 0.4	0.0[c]	81.1 ± 1.3
Schema-in-Prompt	Qwen3-1.7B	5	99.0 ± 0.2	76.6 ± 4.3	98.5 ± 0.3
	xLAM-2-1B[g]	5	98.7 ± 0.3	71.0 ± 4.3	98.5 ± 0.3
	Arch-Function-1.5B[g]	5	99.1 ± 0.1	73.7 ± 4.6	98.6 ± 0.5
	OLMo-2-1B	5	98.9 ± 0.2	44.9 ± 3.9	98.6 ± 0.4
	Llama-3.2-1B	5	99.2 ± 0.2	43.9 ± 8.2	98.4 ± 0.5
	Gemma 3-1B	5	99.2 ± 0.2	29.4 ± 4.6	98.1 ± 0.6
	Qwen3-0.6B	5	98.8 ± 0.3	61.0 ± 4.0	98.3 ± 0.2
	Hammer2.1-0.5B[g]	5	98.8 ± 0.4	46.9 ± 2.0	97.9 ± 0.3
	FunctionGemma-270M	5	97.1 ± 0.4	17.9 ± 2.2	97.1 ± 0.5
	Gemma 3-270M	5	97.5 ± 0.2	15.9 ± 1.9	96.6 ± 0.5
Ablation	Qwen3-1.7B FT, LoRA r=16	3	95.5 ± 0.9	0.0[c]	82.6 ± 0.4
	Qwen3-1.7B SIP, LoRA r=16	5	98.1 ± 0.3	83.2 ± 2.0	96.1 ± 0.6
	FunctionGemma-270M SIP, native format	3	97.4 ± 0.3	24.9 ± 2.7	94.8 ± 0.5

D.4 Out-of-Scope Abstention Decomposition

Out-of-Scope Requests arise in two ways: the requested functionality is absent from the Function Surface, or the function exists but is not offered for the current example. Table 7 splits the second kind by whether the unavailable function is a Seen Function, which has a Functional Token. The grouping comes from the Function Surface partition, not from model predictions, and n counts unique evaluation requests, each evaluated by every model and seed. When the Functional Token arm has a token for the unavailable function it almost never abstains, because its input does not show that the function is unavailable; when it has no token, both representations abstain almost always. The aggregate Out-of-Scope gap between the representations is therefore concentrated in the first group (Figure 7).

Table 6: Behavioral rates (%) averaged over seeds. On Unseen Functions, accuracy requires an exact match and Execution counts any call to an offered function; their difference counts executions that do not match the target. On Out-of-Scope Requests, Over-trigger is the rate of calling a function instead of abstaining.

Representation	Model	Unseen Accuracy	Unseen Execution	OOS Over-trigger
Functional Token	Qwen3-1.7B	0.0	1.4	17.3
	xLAM-2-1B	0.0	1.0	17.2
	Arch-Function-1.5B	0.0	0.9	17.0
	OLMo-2-1B	0.0	1.1	16.7
	Llama-3.2-1B	0.0	0.4	17.4
	Gemma 3-1B	0.0	0.5	17.0
	Qwen3-0.6B	0.0	0.7	17.4
	Hammer2.1-0.5B	0.0	0.7	17.6
	FunctionGemma-270M	0.0	1.2	18.8
	Gemma 3-270M	0.0	1.7	18.9
Schema-in-Prompt	Qwen3-1.7B	76.6	78.7	1.5
	xLAM-2-1B	71.0	73.9	1.5
	Arch-Function-1.5B	73.7	77.3	1.4
	OLMo-2-1B	44.9	46.7	1.4
	Llama-3.2-1B	43.9	46.8	1.6
	Gemma 3-1B	29.4	30.0	1.9
	Qwen3-0.6B	61.0	63.9	1.7
	Hammer2.1-0.5B	46.9	48.5	2.1
	FunctionGemma-270M	17.9	21.0	2.9
	Gemma 3-270M	15.9	19.0	3.4

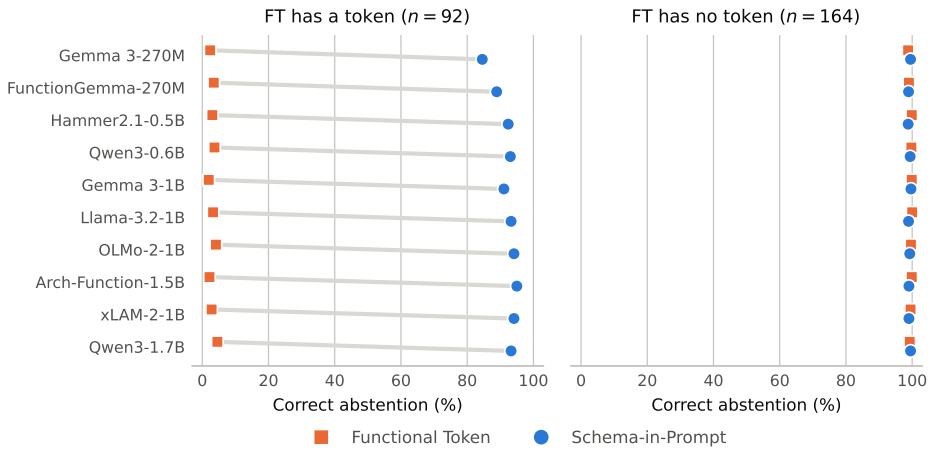


Figure 7: Correct abstention on Out-of-Scope Requests whose function exists but is not offered, per model and averaged over seeds; the rows of Table 7. Left, the unavailable function has a Functional Token; right, it has none. A cell is drawn only when all its seeds exist.

D.5 Offered-Set Functional Token Probe

To test whether the Functional Token arm's abstention failures come from its representation or only from not being told what is offered, we trained a variant whose prompt lists the offered functions as their Functional Tokens, shuffled per example, before the request. An offered Unseen Function contributes nothing to this list, because the arm has no token for it. The probe covers four models:

Table 7: Correct abstention (%) on Out-of-Scope Requests whose function exists but is not offered, split by whether the Functional Token arm has a token for it, averaged over seeds.

Unavailable-function group	Model	FT abstention	SIP abstention
FT has a token ($n = 92$)	Gemma 3-270M	2.4	84.6
	FunctionGemma-270M	3.5	88.9
	Hammer2.1-0.5B	3.0	92.4
	Qwen3-0.6B	3.7	93.0
	Gemma 3-1B	2.0	91.1
	Llama-3.2-1B	3.3	93.3
	OLMo-2-1B	4.1	94.1
	Arch-Function-1.5B	2.2	95.0
	xLAM-2-1B	2.8	94.1
	Qwen3-1.7B	4.6	93.3
FT has no token ($n = 164$)	Gemma 3-270M	98.8	99.5
	FunctionGemma-270M	99.0	98.9
	Hammer2.1-0.5B	99.9	98.8
	Qwen3-0.6B	99.8	99.4
	Gemma 3-1B	99.9	99.6
	Llama-3.2-1B	100.0	98.9
	OLMo-2-1B	99.6	99.3
	Arch-Function-1.5B	99.9	99.0
	xLAM-2-1B	99.5	99.0
	Qwen3-1.7B	99.3	99.5

Gemma 3-270M, FunctionGemma-270M, Qwen3-0.6B and the LoRA Qwen3-1.7B cell. Table 8 shows that the list recovers 20.5 to 41.7% of the Out-of-Scope gap to Schema-in-Prompt.

Table 8: Out-of-Scope accuracy (%) of the Functional Token arm with and without the offered list, against Schema-in-Prompt, and the share of the gap the list recovers.

	Out-of-Scope accuracy			Gap
Model	FT	FT, offered set	SIP	recovered
Gemma 3-270M	81.1	87.3	96.6	40.2%
FunctionGemma-270M	81.1	87.8	97.1	41.7%
Qwen3-0.6B	82.6	87.6	98.3	31.5%
Qwen3-1.7B (LoRA)	82.6	85.4	96.1	20.5%

D.6 Paired Comparisons

Table 9 gives every paired comparison quoted in Section 6. Table 10 gives Unseen accuracy per held-out property group; its intervals are marginal, and because the spread between groups is much larger than the spread between cells, two overlapping marginal intervals are not evidence that two cells are equal. Only the paired intervals compare cells. One comparison was registered before its data were seen. Arch-Function-1.5B and xLAM-2-1B share a base model and differ only in their vendors' function-calling fine-tuning. After identical fine-tuning on our corpus, the two are not separable on Unseen Functions (xLAM minus Arch-Function, -1.9 points $[-8.5, +4.4]$, group-weighted, 5 seeds each; both trained under the guard), and their Seen and Out-of-Scope accuracies differ by little in either representation (Table 5). On this one pair, we detect no difference between the vendor recipes after our fine-tuning.

D.7 Adaptation and Prompt-Format Ablations

Two ablations test choices that could otherwise confound the main grid (Table 11). The first adapts Qwen3-1.7B's Schema-in-Prompt arm with LoRA instead of full fine-tuning. Its row-weighted

Table 9: Paired differences in Unseen accuracy, group-weighted, by paired bootstrap: the crossed interval, and the per-group scheme as a sensitivity check. n is the number of seeds per cell.

Comparison	n	Δ	95% CI	Per-group CI	Verdict
qwen06b-sip − gemma1b-sip	5/5	$+0.218$	$[+0.089, +0.367]$	$[+0.077, +0.382]$	separable
qwen06b-sip − llama1b-sip	5/5	$+0.108$	$[+0.016, +0.218]$	$[-0.004, +0.246]$	unresolved ($p \approx 0.02$ crossed)
qwen06b-sip − olmo1b-sip	5/5	$+0.084$	$[+0.005, +0.195]$	$[-0.013, +0.207]$	unresolved ($p \approx 0.03$ crossed)
olmo1b-sip − gemma1b-sip	5/5	$+0.134$	$[+0.046, +0.228]$	$[+0.025, +0.248]$	separable
llama1b-sip − gemma1b-sip	5/5	$+0.110$	$[-0.015, +0.241]$	$[-0.039, +0.263]$	not separable ($p \approx 0.08$)
llama1b-sip − olmo1b-sip	5/5	-0.024	$[-0.122, +0.075]$	$[-0.157, +0.097]$	not separable ($p \approx 0.61$)
qwen17b-sip-fullft − qwen06b-sip	5/5	$+0.123$	$[+0.063, +0.192]$	$[+0.053, +0.201]$	separable
qwen17b-sip-fullft − qwen17b-sip	5/5	-0.038	$[-0.118, +0.031]$	$[-0.120, +0.037]$	not separable ($p \approx 0.31$)
gemma1b-sip − gemma270m-sip	5/5	$+0.185$	$[+0.049, +0.344]$	$[+0.033, +0.355]$	separable
gemma1b-sip − fg270m-sip	5/5	$+0.158$	$[+0.043, +0.288]$	$[+0.028, +0.297]$	separable
qwen06b-sip − fg270m-sip	5/5	$+0.376$	$[+0.205, +0.534]$	$[+0.203, +0.541]$	separable
fg270m-sip − gemma270m-sip	5/5	$+0.027$	$[-0.022, +0.100]$	$[-0.038, +0.117]$	not separable ($p \approx 0.37$)
archfn15b-sip − qwen17b-sip-fullft	5/5	-0.021	$[-0.075, +0.034]$	$[-0.089, +0.042]$	not separable ($p \approx 0.44$)
archfn15b-sip − olmo1b-sip	5/5	$+0.187$	$[+0.074, +0.315]$	$[+0.064, +0.325]$	separable
archfn15b-sip − llama1b-sip	5/5	$+0.211$	$[+0.083, +0.354]$	$[+0.067, +0.377]$	separable
hammer05b-sip − qwen06b-sip	5/5	-0.097	$[-0.201, -0.014]$	$[-0.218, +0.003]$	unresolved ($p \approx 0.02$ crossed)
hammer05b-sip − gemma1b-sip	5/5	$+0.121$	$[+0.017, +0.224]$	$[-0.003, +0.245]$	unresolved ($p \approx 0.02$ crossed)
xlam15b-sip − archfn15b-sip	5/5	-0.019	$[-0.085, +0.044]$	$[-0.095, +0.056]$	not separable ($p \approx 0.56$)

Table 10: Unseen accuracy of each Schema-in-Prompt cell per held-out property group, with a marginal 95% cluster interval. These intervals must not be used to compare two cells.

Held-out property group	gemma270m-sip	fg270m-sip	hammer05b-sip	qwen06b-sip	gemma1b-sip	llama1b-sip	olmo1b-sip	archfn15b-sip	xlam15b-sip	qwen17b-sip-fullft
CRUISE_CONTROL	0.148	0.142	0.677	0.712	0.430	0.412	0.628	0.950	0.842	0.965
EV_CHARGING	0.119	0.177	0.628	0.693	0.420	0.559	0.647	0.751	0.705	0.884
FOG_LIGHTS	0.232	0.425	0.871	0.893	0.754	0.968	0.907	0.911	0.971	0.968
HVAC_DEFROST	0.278	0.230	0.370	0.386	0.269	0.368	0.382	0.444	0.453	0.395
MIRROR_POSITION	0.041	0.019	0.090	0.141	0.101	0.051	0.075	0.368	0.234	0.300
SEAT_LUMBAR	0.088	0.126	0.381	0.702	0.056	0.442	0.287	0.793	0.865	0.841
STEERING_WHEEL_CLIMATE	0.218	0.204	0.546	0.826	0.579	0.775	0.775	0.960	0.947	0.972
WINDOW_LOCK	0.971	0.987	0.980	0.967	0.967	0.881	0.945	0.963	0.972	0.982
Equal-weighted mean	0.262	0.289	0.568	0.665	0.447	0.557	0.581	0.768	0.749	0.789
95% cluster interval	[0.120, 0.479]	[0.132, 0.510]	[0.378, 0.751]	[0.477, 0.825]	[0.253, 0.651]	[0.357, 0.748]	[0.374, 0.768]	[0.603, 0.907]	[0.562, 0.906]	[0.589, 0.947]
Seed half-range	±0.022	±0.028	±0.021	±0.053	±0.060	±0.104	±0.053	±0.061	±0.053	±0.051

Unseen accuracy is higher, but the group-weighted paired difference, full minus LoRA, is -3.8 points $[-11.8, +3.1]$ and not separable (Table 9); the full fine-tune attempts fewer Unseen requests and is at least as accurate when it attempts. The second trains FunctionGemma-270M in its native tool-calling format instead of the uniform JSON schema format used for every other model. It raises Unseen accuracy from 17.9% to 24.9% and lowers Out-of-Scope accuracy from 97.1% to 94.8%; the main-grid comparison with Gemma 3-270M is made in the uniform format, which neither model was built for.

Table 11: Schema-in-Prompt ablations beside the main-grid cell each one varies. Accuracy (%) averaged over seeds.

Model	SIP configuration	Seen	Unseen	Out-of-Scope
Qwen3-1.7B	full (main grid)	99.0	76.6	98.5
	LoRA r=16 (adaptation ablation)	98.1	83.2	96.1
FunctionGemma-270M	Uniform JSON Schema (main grid)	97.1	17.9	97.1
	Native tool-calling format	97.4	24.9	94.8

D.8 Unseen Decomposition

Table 12 factors each Schema-in-Prompt cell's Unseen accuracy into its attempt rate and its accuracy given an attempt. For Qwen3-0.6B, Llama-3.2-1B and OLMo-2-1B, equivalence of accuracy given an attempt within three points cannot be established at 5 seeds, so Table 13 reports the paired differences in accuracy given an attempt. These differences are small next to the differences in attempt rate, which is the sense in which Section 6.2 attributes most of each gap to the attempt rate.

Across the three models, Unseen accuracy spans 17.1 points; the attempt rate accounts for 0.87 of the log gap from Qwen3-0.6B to OLMo-2-1B and 0.96 of the gap to Llama-3.2-1B. The 270M

models attempt 32.2% and 30.8% of Unseen requests and are right on 49.3% and 58.0% of their attempts. The attempt rate is not a pure measure of schema reading. Abstention is the correct output on 25% of training rows, so a model that learns to abstain on unfamiliar requests may also decline Unseen requests it could have answered; the share is the same in every cell, so its effect is untested.

Table 12: Unseen accuracy (%) of each Schema-in-Prompt cell, factored into attempt rate and accuracy given an attempt.

Model	Accuracy	Attempted	Accuracy | attempted
Gemma 3-270M	15.9	32.2	49.3
FunctionGemma-270M	17.9	30.8	58.0
Hammer2.1-0.5B	46.9	50.4	93.0
Qwen3-0.6B	61.0	66.0	92.4
Gemma 3-1B	29.4	37.4	78.6
Llama-3.2-1B	43.9	48.1	91.2
OLMo-2-1B	44.9	50.5	88.8
Arch-Function-1.5B	73.7	78.1	94.4
xLAM-2-1B	71.0	74.6	95.2
Qwen3-1.7B	76.6	80.0	95.8

Table 13: Paired differences in Unseen accuracy given an attempt for the three mid-ladder cells, same bootstrap as Table 9, attempts pooled.

Comparison	n	Δ	95% CI	Per-group CI	Verdict
qwen06b-sip − olmo1b-sip	5/5	$+0.035$	$[+0.001, +0.095]$	$[-0.024, +0.119]$	unresolved ($p \approx 0.05$ crossed)
qwen06b-sip − llama1b-sip	5/5	$+0.011$	$[-0.046, +0.072]$	$[-0.062, +0.095]$	not separable ($p \approx 0.90$)
llama1b-sip − olmo1b-sip	5/5	$+0.024$	$[-0.031, +0.096]$	$[-0.068, +0.124]$	not separable ($p \approx 0.30$)

D.9 Conformal Calibration

The calibrated rule uses a forced decoder: at the first token where a call and the abstention differ, it forces the call branch, decodes the rest of the call greedily, and commits when the sequence log-odds of that call against the abstention exceed the threshold. On almost every request its call-or-abstain decision at a threshold of zero matches the decision behind the main tables; the exceptions are requests on which greedy decoding chooses neither branch token. The exact task-level score $p_{\mathrm{SIP}}(f \mid x, \mathcal{S}(O))$ cannot be computed, because Schema-in-Prompt targets serialize the arguments before the function name, so the score would sum over every argument string; the sequence log-odds of the greedy completion of the call branch is used instead, and by Theorem 4.1(a) its first term lower-bounds the log-probability of the committed function.

The primary loss is function-level: a committed call to a function other than the target, any committed call on an Out-of-Scope Request, and any committed output that does not parse. For each cell and seed, 2,000 random splits of the 3,214 evaluation requests, which are 66.5% Unseen, 17.0% Seen and 16.6% Out-of-Scope, give calibration sets of 1,000 requests and disjoint test sets, with α from 0.01 to 0.20. Group splits assign whole functions and unsupported request types to one side, with about 1,000 calibration requests. The correction term in the threshold matters only for small calibration sets: with 50 calibration requests, the uncorrected threshold (the smallest one whose empirical risk is at most α) exceeds α by up to 1.9 points over the range of α, while the corrected one does not (at $\alpha = 0.05$ in Table 14). The comparison of the two representations calibrates within Seen and Out-of-Scope requests only, with 500 calibration requests, and the changed-request experiment calibrates on those requests and tests on the rest. Tables 14 to 17 give the per-cell results.

At $\alpha = 0.05$ (Table 15), when a model's own error rate is below α, the calibrated threshold commits on requests on which the model abstained: Qwen3-1.7B's rate of right-function calls on Unseen requests rises from 79.3% to 95.2% while its Out-of-Scope commit rate rises from 1.5% to 19.2%, and the largest Out-of-Scope commit rate among Schema-in-Prompt cells reaches 22.9%. For the strongest models, calibration thus raises the confident-error rate up to the budget. When the own error rate is above α, as for the 270M models, calibration abstains more and that rate falls. These

rates are function level, so they exceed the exact-match accuracies of Table 1. Calibrating on the exact-call loss instead of the function-level one keeps the exact-call risk at 5.0% or below at $\alpha = 0.05$ on random splits. Calibrating without Unseen requests and testing on all requests breaks the bound for Functional Token once α reaches 0.10 to 0.13 and for every Schema-in-Prompt cell except Qwen3-1.7B, xLAM-2-1B and Arch-Function-1.5B, most of them from the smallest α; these three hold because their Unseen calls are about as reliable as their Seen ones (Table 17). This experiment changes which functions are requested while the Function Surface stays fixed; an update that adds functions would change both. The calibrated thresholds are in units of the batched 16-bit scores and do not carry over to batch-1 or 4-bit scoring, and the sequence score needs an extra decoding pass and an abstention scoring pass that no Device Proxy figure measures.

Table 14: Mean test confident-error risk (%) at $\alpha = 0.05$: random splits with the corrected threshold and with the uncorrected one at small n, and splits by whole functions and request types. Rates in percent, means over 5 training seeds; $\pm$ is the standard deviation over seeds. Sequence log-odds score, function-level loss, corrected threshold unless stated. The eval pool holds 3,214 requests: 66.5% Unseen, 17.0% Seen and 16.6% Out-of-Scope, and α is a budget over that mix. "Own" is the model's uncalibrated decision. [s]Structural: Functional Token has no token for an Unseen Function. [§]Seeds 0 and 1 of this cell come from a later training run than the predictions behind the main tables. A cell fails from the first α at which its mean test risk exceeds α by more than 0.5 points.

		Row splits		Group splits	
Rep.	Model	corrected, n=1,000	uncorrected, n=50	mean	90th pct.
FT	Qwen3-1.7B	5.0	5.8	7.9 ± 0.6	16.4
FT	xLAM-2-1B	5.0	5.8	7.5 ± 0.6	14.9
FT	Arch-Function-1.5B	5.0	5.8	7.3 ± 0.2	15.1
FT	OLMo-2-1B	5.0	5.8	7.6 ± 0.4	16.4
FT	Llama-3.2-1B	5.0	5.9	7.9 ± 0.3	16.9
FT	Gemma 3-1B	5.0	5.8	8.6 ± 0.6	18.4
FT	Qwen3-0.6B	5.0	5.9	7.4 ± 0.6	13.9
FT	Hammer2.1-0.5B	5.0	5.8	7.5 ± 0.6	14.8
FT	FunctionGemma-270M	5.0	5.8	7.0 ± 0.3	14.5
FT	Gemma 3-270M	5.0	5.8	7.7 ± 0.7	15.6
SIP	Qwen3-1.7B	5.0	5.9	5.8 ± 0.1	9.3
SIP	xLAM-2-1B	5.0	5.9	5.8 ± 0.2	9.7
SIP	Arch-Function-1.5B	5.0	5.9	5.7 ± 0.2	8.6
SIP	OLMo-2-1B	5.0	5.9	5.8 ± 0.5	10.6
SIP	Llama-3.2-1B	5.0	5.9	5.7 ± 0.2	10.2
SIP	Gemma 3-1B	5.0	5.9	6.4 ± 0.8	12.0
SIP	Qwen3-0.6B	5.0	5.9	6.2 ± 0.8	10.9
SIP	Hammer2.1-0.5B	5.0	5.9	6.0 ± 0.4	11.2
SIP	FunctionGemma-270M[§]	5.0	5.9	6.7 ± 0.4	13.6
SIP	Gemma 3-270M	5.0	5.8	7.0 ± 0.4	14.6

D.10 Device Proxy Measurements

Accuracy is measured on the 16-bit checkpoints, and Appendix D.11 reports how the 4-bit exports change it. All latency and memory figures are Device Proxy measurements, because we have no in-car hardware; they are taken on a Xeon Platinum 8580 server CPU with four threads running llama.cpp on each cell's Q4_K_M export. *Time to first call* is the elapsed time until the output first forms a parseable Vehicle Function Call or a readable abstention, since an abstention is also an answer the driver receives. Each figure is the median over three runs, with the range in parentheses (Table 18). Run-to-run noise on the short Functional Token prompts is large relative to their differences, so Functional Token cells of similar size are not ranked against each other, and peak resident memory is reported but not used to rank cells. Figure 8 compares the time to first call of the two representations of each model.

Table 19 gives costs that do not depend on the host, counted from each export's tensor table: the bytes read per generated token, the attention cache at the median prompt length, and the prefill

Table 15: What calibration at $\alpha = 0.05$ costs and which way it moves each cell: the model's own risk, the calibrated threshold, the rate of right-function calls on Unseen requests (function level, arguments not checked) and the Out-of-Scope commit rate, before and after calibration.

		Own		Unseen correct		OOS commit	
Rep.	Model	risk	$\hat{\gamma}$	own	calibrated	own	calibrated
FT	Qwen3-1.7B	4.7	-0.3 ± 0.7	0.0^{s}	0.0^{s}	17.2	17.2
FT	xLAM-2-1B	4.1	-1.0 ± 0.9	0.0^{s}	0.0^{s}	17.1	17.5
FT	Arch-Function-1.5B	4.0	-1.1 ± 0.5	0.0^{s}	0.0^{s}	17.0	17.5
FT	OLMo-2-1B	3.9	-1.1 ± 0.3	0.0^{s}	0.0^{s}	16.6	17.1
FT	Llama-3.2-1B	3.4	-1.9 ± 0.5	0.0^{s}	0.0^{s}	17.4	18.9
FT	Gemma 3-1B	4.0	-1.3 ± 1.0	0.0^{s}	0.0^{s}	17.0	17.4
FT	Qwen3-0.6B	4.7	-0.5 ± 0.9	0.0^{s}	0.0^{s}	17.4	17.5
FT	Hammer2.1-0.5B	4.0	-1.4 ± 0.8	0.0^{s}	0.0^{s}	17.5	18.5
FT	FunctionGemma-270M	5.5	$+0.2 \pm 0.5$	0.0^{s}	0.0^{s}	18.7	18.2
FT	Gemma 3-270M	6.2	$+0.5 \pm 0.9$	0.0^{s}	0.0^{s}	18.9	17.8
SIP	Qwen3-1.7B	0.8	-13.9 ± 0.9	79.3	95.2 ± 1.1	1.5	19.2
SIP	xLAM-2-1B	0.7	-11.3 ± 0.5	74.2	95.0 ± 1.5	1.6	16.8
SIP	Arch-Function-1.5B	0.6	-11.6 ± 0.5	77.5	97.4 ± 0.4	1.4	22.9
SIP	OLMo-2-1B	2.6	-4.9 ± 1.1	47.2	66.4 ± 4.7	1.4	3.9
SIP	Llama-3.2-1B	1.4	-8.4 ± 2.4	46.7	74.3 ± 7.5	1.6	8.3
SIP	Gemma 3-1B	5.1	-0.6 ± 1.7	30.4	32.0 ± 5.1	1.8	2.1
SIP	Qwen3-0.6B	2.2	-5.6 ± 2.0	63.5	78.9 ± 9.3	1.6	5.6
SIP	Hammer2.1-0.5B	1.9	-7.1 ± 2.5	48.4	68.1 ± 6.4	2.1	8.6
SIP	FunctionGemma-270M[§]	8.3	$+3.7 \pm 1.0$	19.0	12.5 ± 1.7	3.0	1.6
SIP	Gemma 3-270M	10.9	$+3.5 \pm 1.3$	17.2	10.7 ± 1.7	3.5	1.8

Table 16: Both representations under one calibration, on Seen and Out-of-Scope requests only, at $\alpha = 0.02$: the rate of right-function calls on Seen requests (function level) before and after calibration, and over-triggering on unavailable functions that have a Functional Token.

	Seen correct, own		Seen correct, α=0.02		Token-group over-trigger	
Model	FT	SIP	FT	SIP	FT	SIP
Gemma 3-270M	98.7	98.6	42.7 ± 7.8	98.6 ± 0.2	22.0	16.1
FunctionGemma-270M[§]	98.5	98.3	41.3 ± 4.0	98.3 ± 0.4	21.9	12.8
Hammer2.1-0.5B	98.6	99.3	42.7 ± 2.4	99.6 ± 0.1	22.4	12.9
Qwen3-0.6B	98.5	99.5	46.1 ± 2.6	99.8 ± 0.1	23.0	13.7
Gemma 3-1B	98.6	99.6	45.9 ± 3.8	99.7 ± 0.2	23.0	16.4
Llama-3.2-1B	98.7	99.8	46.0 ± 4.9	99.8 ± 0.1	22.3	14.9
OLMo-2-1B	98.4	99.6	42.5 ± 3.4	99.7 ± 0.1	22.8	14.5
Arch-Function-1.5B	98.8	99.9	42.9 ± 1.5	100.0 ± 0.1	22.7	12.0
xLAM-2-1B	98.5	99.6	43.5 ± 2.3	99.7 ± 0.2	22.9	13.8
Qwen3-1.7B	98.3	99.8	44.5 ± 3.6	99.8 ± 0.1	23.0	14.2

operations. The read rates these imply on the Device Proxy are consistent with decoding being limited by memory traffic, although the host's attainable bandwidth is not part of the measurement. Table 20 gives Schema-in-Prompt's time to first call with the processed schema prefix cached and reused, and the accuracy change the cache causes, none of it significant.

D.11 Accuracy of the 4-Bit Exports

Table 21 compares each cell's Q4_K_M export, the file the Device Proxy times, with its 16-bit checkpoint over the seeds both exist for; a superscript k/n counts the seeds on which the change is significant. The exports are scored with llama.cpp's GPU kernels; re-scoring two files with the CPU kernels the Device Proxy uses reproduced the direction and size of each effect, but a fragile 4-bit

Table 17: Calibrating without Unseen requests and testing on all requests: the largest excess of mean risk over α, the α from which a cell fails, and the risk at $\alpha = 0.10$ with the part from Unseen rows.

		Max excess	Fails	Risk at α=0.10	
Rep.	Model	(points)	from α	total	from Unseen
FT	Qwen3-1.7B	+25.4	0.10	10.9	8.8
FT	xLAM-2-1B	+26.2	0.11	9.2	7.0
FT	Arch-Function-1.5B	+22.6	0.11	9.8	7.7
FT	OLMo-2-1B	+20.3	0.10	11.2	9.1
FT	Llama-3.2-1B	+16.8	0.13	5.7	3.6
FT	Gemma 3-1B	+15.0	0.11	8.7	6.6
FT	Qwen3-0.6B	+14.6	0.12	7.8	5.7
FT	Hammer2.1-0.5B	+16.4	0.13	6.5	4.4
FT	FunctionGemma-270M	+22.0	0.12	7.4	5.3
FT	Gemma 3-270M	+19.4	0.11	8.5	6.4
SIP	Qwen3-1.7B	−0.1	never	4.3	2.2
SIP	xLAM-2-1B	−0.4	never	5.1	3.0
SIP	Arch-Function-1.5B	−0.2	never	3.5	1.4
SIP	OLMo-2-1B	+4.4	0.01	12.7	10.6
SIP	Llama-3.2-1B	+1.9	0.01	10.7	8.6
SIP	Gemma 3-1B	+8.7	0.01	18.3	16.2
SIP	Qwen3-0.6B	+3.3	0.01	10.6	8.5
SIP	Hammer2.1-0.5B	+1.8	0.02	11.3	9.2
SIP	FunctionGemma-270M[§]	+20.2	0.01	30.2	28.1
SIP	Gemma 3-270M	+23.9	0.01	33.8	31.7

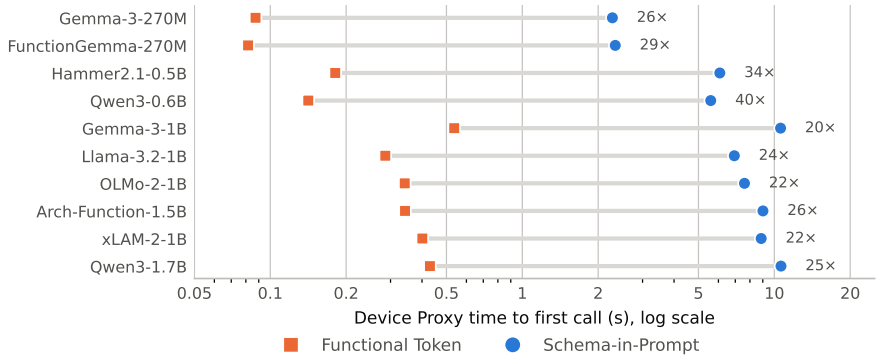


Figure 8: Device Proxy time to first call of each model's Functional Token and Schema-in-Prompt cells, median over three runs on a server CPU at four threads, log scale; the ratio is printed at the right. These are proxy figures: no claim is made about latency in a vehicle.

file's exact accuracy can differ by several points between the two. Seeds 0 and 1 of FunctionGemma-270M's Schema-in-Prompt arm are excluded from this pairing, because their exports and their 16-bit predictions come from different training runs. Tables 22 to 24 step one seed of each cell from 16-bit down to Q4_K_M, one table per split; with one seed, a drop at a given level is an observation rather than an estimate.

The 270M Functional Token cells lose 14.0 to 18.6 points on Seen Functions, mostly by abstaining more, whereas Schema-in-Prompt cells abstain less, which turns some Unseen abstentions into

Table 18: Device Proxy cost per cell, median (range) over three runs; lower is better for time and memory. These are proxy figures: no claim is made about latency in a vehicle.

Representation	Model	Prompt tokens	Time to first call (ms)	Peak RSS (MB)
Functional Token	Qwen3-1.7B	14	430 (410–465)	2701 (2699–2701)
	xLAM-2-1B	14	401 (379–469)	2093 (2092–2093)
	Arch-Function-1.5B	14	342 (337–373)	2093 (2092–2093)
	OLMo-2-1B	14	342 (325–420)	2163 (2163–2163)
	Llama-3.2-1B	15	286 (263–328)	1558 (1557–1558)
	Gemma 3-1B	16	537 (509–588)	1102 (1099–1102)
	Qwen3-0.6B	14	141 (124–143)	1216 (1215–1338)
	Hammer2.1-0.5B	14	181 (171–181)	586 (586–586)
	FunctionGemma-270M	16	82 (77–90)	408 (407–411)
	Gemma 3-270M	16	87 (64–88)	407 (405–407)
Schema-in-Prompt	Qwen3-1.7B	1,799	10627 (10446–10977)	6516 (6516–6516)
	xLAM-2-1B	1,799	8867 (8301–8875)	3106 (3106–3106)
	Arch-Function-1.5B	1,799	9012 (8813–9892)	3106 (3106–3106)
	OLMo-2-1B	1,784	7612 (7524–7965)	6499 (6499–6500)
	Llama-3.2-1B	1,785	6937 (6724–7284)	2907 (2907–2907)
	Gemma 3-1B	1,921	10587 (10582–11154)	2336 (2336–2337)
	Qwen3-0.6B	1,799	5593 (5254–5666)	5126 (5126–5126)
	Hammer2.1-0.5B	1,799	6077 (6030–6176)	1033 (1033–1033)
	FunctionGemma-270M	1,921	2338 (2290–2349)	1208 (1208–1209)
	Gemma 3-270M	1,921	2277 (2248–2341)	1208 (1208–1209)

Table 19: Hardware-independent cost per cell, counted from its Q4_K_M export. The last column is derived from a Device Proxy figure.

Cell	File (MB)	Read/token (MB)	Head share	Prompt tokens	Cache (MB)	Prefill (GFLOP)	Device Proxy read (GB/s)
gemma270m-ft	253	247	72%	16	0.3	3.6	29.3
gemma270m-sip	253	247	72%	1,921	13.8	460.9	32.1
fg270m-ft	253	247	72%	16	0.3	3.6	28.7
fg270m-sip	253	247	72%	1,921	13.8	460.9	30.6
hammer05b-ft	398	392	37%	14	0.2	10.3	22.8
hammer05b-sip	398	392	37%	1,799	22.1	1427.3	17.9
qwen06b-ft	397	391	33%	14	1.6	12.7	29.2
qwen06b-sip	397	391	33%	1,799	206.3	1956.5	14.9
gemma1b-ft	806	800	40%	16	0.4	23.0	16.4
gemma1b-sip	806	800	40%	1,921	19.4	2789.0	23.9
llama1b-ft	808	800	27%	15	0.5	29.7	27.0
llama1b-sip	808	800	27%	1,785	58.5	3683.6	22.2
olmo1b-ft	936	816	21%	14	1.8	30.5	20.8
olmo1b-sip	936	816	21%	1,784	233.8	4040.7	19.0
archfn15b-ft	986	980	20%	14	0.4	37.2	23.9
archfn15b-sip	986	980	20%	1,799	51.6	4993.6	21.2
xlam15b-ft	986	980	20%	14	0.4	37.2	21.1
xlam15b-sip	986	980	20%	1,799	51.6	4993.6	22.6
qwen17b-ft	1107	1101	23%	14	1.6	40.1	26.2
qwen17b-sip	1107	1101	23%	1,799	206.3	5443.1	20.2
qwen17b-ft-fullft	1107	1101	23%	14	1.6	40.1	23.1
qwen17b-sip-fullft	1107	1101	23%	1,799	206.3	5443.1	19.0

correct calls and more of them into wrong ones. The two LoRA-trained Qwen3-1.7B ablation cells (Appendix D.7) also lose accuracy, the Functional Token one falling from 95.5% to 42.3% on Seen.

Table 20: Device Proxy time to first call with the schema prefix cached, against uncached in the same run, and the accuracy change (points) the cache causes. The Functional Token column comes from a different run, so a ratio to it is approximate.

	Device Proxy time to first call (ms)				Accuracy under the cache				
Cell	Uncached	Cached	Schema, once	FT	Seeds	Seen	Unseen	OOS	Changed
gemma270m-sip	2,207	221	1,990	87	3	-0.4	-0.1	-0.1	7.4%
fg270m-sip	2,230	248	2,041	82	3	-0.6	-0.0	-0.1	7.2%
hammer05b-sip	6,122	613	5,593	181	3	-0.1	-0.4	+0.1	1.6%
qwen06b-sip	5,338	1,127	4,445	141	3	+0.0	-0.4	+0.0	2.7%
gemma1b-sip	10,750	882	9,809	537	3	+0.1	-0.0	+0.1	1.6%
llama1b-sip	6,712	1,018	5,862	286	3	-0.1	+0.4	-0.1	1.3%
olmo1b-sip	7,414	1,296	6,096	342	2	+0.1	+0.1	+0.0	1.9%
archfn15b-sip	8,372	1,067	7,301	342	3	-0.1	+0.1	+0.1	1.3%
xlam15b-sip	8,628	1,091	7,570	401	3	+0.0	+0.3	-0.1	1.6%
qwen17b-sip	10,267	1,733	8,883	–	3	+0.1	+0.2	-0.0	2.7%
qwen17b-sip-fullft	10,394	1,748	8,894	430	1	+0.4	-0.0	-0.4	1.3%

Table 21: Accuracy of the Q4_K_M export beside its 16-bit checkpoint, over the seeds both exist for. A superscript k/n marks the number of seeds whose change is significant.

		Seen		Unseen		Out-of-Scope	
Cell	Seeds	16-bit	Q4_K_M	16-bit	Q4_K_M	16-bit	Q4_K_M
gemma270m-ft	5	0.974	$0.834^{5/5}$	0.000	0.000	0.811	$0.843^{4/5}$
gemma270m-sip	5	0.975	$0.961^{3/5}$	0.159	$0.228^{5/5}$	0.966	$0.921^{5/5}$
fg270m-ft	5	0.970	$0.784^{5/5}$	0.000	0.000	0.811	$0.853^{4/5}$
fg270m-sip	3	0.968	$0.948^{2/3}$	0.173	$0.220^{3/3}$	0.969	$0.951^{3/3}$
hammer05b-ft	5	0.977	$0.968^{1/5}$	0.000	0.000	0.824	0.827
hammer05b-sip	5	0.988	$0.977^{2/5}$	0.469	$0.422^{4/5}$	0.979	0.979
qwen06b-ft	5	0.975	$0.943^{4/5}$	0.000	0.000	0.826	$0.836^{2/5}$
qwen06b-sip	5	0.988	$0.979^{1/5}$	0.610	$0.644^{4/5}$	0.983	0.979
gemma1b-ft	5	0.980	0.978	0.000	0.000	0.830	0.831
gemma1b-sip	5	0.992	0.989	0.294	$0.318^{3/5}$	0.981	0.979
llama1b-ft	5	0.981	$0.978^{1/5}$	0.000	0.000	0.826	$0.830^{1/5}$
llama1b-sip	5	0.992	0.993	0.439	$0.428^{3/5}$	0.984	0.983
olmo1b-ft	4	0.977	0.969	0.000	0.000	0.833	$0.838^{1/4}$
olmo1b-sip	4	0.989	0.987	0.450	$0.474^{2/4}$	0.985	0.983
archfn15b-ft	5	0.981	$0.973^{2/5}$	0.000	0.000	0.830	0.835
archfn15b-sip	5	0.991	0.989	0.737	$0.698^{5/5}$	0.986	0.989
xlam15b-ft	5	0.975	$0.961^{3/5}$	0.000	0.000	0.828	$0.834^{1/5}$
xlam15b-sip	5	0.987	0.985	0.710	$0.722^{2/5}$	0.985	0.984
qwen17b-ft	3	0.955	$0.423^{3/3}$	0.000	0.000	0.826	$0.907^{2/3}$
qwen17b-sip	5	0.981	$0.921^{5/5}$	0.832	$0.790^{5/5}$	0.961	$0.965^{1/5}$
qwen17b-ft-fullft	5	0.974	$0.948^{5/5}$	0.000	0.000	0.827	$0.836^{1/5}$
qwen17b-sip-fullft	3	0.990	0.985	0.761	$0.784^{2/3}$	0.984	0.983

Table 22: Seen accuracy from 16-bit down to Q4_K_M at seed 0, from fresh conversions of each checkpoint. One seed only.

Cell	16-bit	BF16	Q8_0	Q6_K	Q5_K_M	Q4_K_M
gemma270m-ft	0.976	0.976	0.969	0.971	0.881	0.859
gemma270m-sip	0.978	0.978	0.978	0.978	0.976	0.965
fg270m-ft	0.967	0.967	0.967	0.971	0.903	0.763
hammer05b-ft	0.985	0.983	0.980	0.987	0.974	0.967
hammer05b-sip	0.991	0.991	0.993	0.989	0.985	0.974
qwen06b-ft	0.978	0.974	0.974	0.969	0.969	0.912
qwen06b-sip	0.983	0.985	0.985	0.978	0.985	0.974
gemma1b-ft	0.983	0.983	0.983	0.982	0.978	0.982
gemma1b-sip	0.993	0.993	0.994	0.993	0.991	0.993
llama1b-ft	0.983	0.983	0.983	0.985	0.985	0.983
llama1b-sip	0.993	0.993	0.991	0.991	0.991	0.994
olmo1b-ft	0.974	0.974	0.974	0.976	0.978	0.965
olmo1b-sip	0.989	0.989	0.989	0.991	0.991	0.987
archfn15b-ft	0.976	0.980	0.980	0.980	0.976	0.976
archfn15b-sip	0.991	0.989	0.985	0.987	0.985	0.987
xlam15b-ft	0.976	0.976	0.974	0.976	0.978	0.974
xlam15b-sip	0.989	0.989	0.989	0.987	0.989	0.987
qwen17b-ft†	0.947	0.943	0.938	0.923	0.798	0.316
qwen17b-sip†	0.978	0.980	0.980	0.978	0.969	0.925
qwen17b-ft-fullft	0.974	0.978	0.972	0.969	0.969	0.945
qwen17b-sip-fullft	0.991	0.991	0.991	0.987	0.987	–

Table 23: Unseen accuracy from 16-bit down to Q4_K_M at seed 0, Schema-in-Prompt cells only (Functional Token cells score 0 by construction at every level). One seed only.

Cell	16-bit	BF16	Q8_0	Q6_K	Q5_K_M	Q4_K_M
gemma270m-sip	0.176	0.173	0.181	0.213	0.202	0.202
hammer05b-sip	0.491	0.490	0.486	0.478	0.500	0.438
qwen06b-sip	0.544	0.545	0.534	0.529	0.574	0.565
gemma1b-sip	0.363	0.365	0.362	0.360	0.351	0.397
llama1b-sip	0.580	0.583	0.580	0.530	0.548	0.582
olmo1b-sip	0.504	0.507	0.503	0.494	0.488	0.555
archfn15b-sip	0.672	0.669	0.665	0.656	0.648	0.603
xlam15b-sip	0.675	0.680	0.669	0.679	0.699	0.685
qwen17b-sip†	0.853	0.859	0.857	0.855	0.856	0.824
qwen17b-sip-fullft	0.743	0.743	0.737	0.729	0.756	–

Table 24: Out-of-Scope accuracy (correct abstention) from 16-bit down to Q4_K_M at seed 0. One seed only.

Cell	16-bit	BF16	Q8_0	Q6_K	Q5_K_M	Q4_K_M
gemma270m-ft	0.827	0.827	0.829	0.829	0.848	0.852
gemma270m-sip	0.968	0.968	0.968	0.959	0.957	0.932
fg270m-ft	0.811	0.812	0.818	0.814	0.820	0.878
hammer05b-ft	0.829	0.829	0.835	0.831	0.835	0.835
hammer05b-sip	0.979	0.977	0.977	0.977	0.979	0.979
qwen06b-ft	0.835	0.835	0.837	0.837	0.837	0.859
qwen06b-sip	0.983	0.983	0.983	0.983	0.981	0.979
gemma1b-ft	0.827	0.827	0.827	0.827	0.827	0.829
gemma1b-sip	0.981	0.981	0.983	0.985	0.981	0.977
llama1b-ft	0.814	0.814	0.816	0.816	0.816	0.816
llama1b-sip	0.983	0.983	0.983	0.989	0.985	0.979
olmo1b-ft	0.829	0.831	0.831	0.831	0.831	0.837
olmo1b-sip	0.987	0.987	0.987	0.987	0.987	0.981
archfn15b-ft	0.826	0.826	0.826	0.833	0.833	0.831
archfn15b-sip	0.989	0.989	0.989	0.989	0.987	0.989
xlam15b-ft	0.829	0.827	0.829	0.826	0.824	0.827
xlam15b-sip	0.989	0.989	0.991	0.989	0.987	0.989
qwen17b-ft†	0.824	0.820	0.820	0.827	0.854	0.953
qwen17b-sip†	0.955	0.951	0.947	0.953	0.940	0.953
qwen17b-ft-fullft	0.835	0.833	0.833	0.835	0.839	0.839
qwen17b-sip-fullft	0.987	0.987	0.987	0.985	0.985	–