Sabaki : un mélange d'experts organisationnel pour des décisions typées sur CPU

Fox3000foxy
Chercheur indépendant
fox3000foxy@users.noreply.github.com

***Résumé*—Les grands modèles de langage sont couramment invoqués pour des tâches qui sont, au fond, de la classification : router ce ticket, scorer ce prospect, signaler ce message. Sabaki remplace l'appel au modèle unique par un mélange d'experts *organisationnel* : un routeur sensible aux mots-clés et à la langue dispatche chaque entrée vers un petit plugin classifieur entraîné et versionné indépendamment (nano TF-IDF, < 1 ms ; mini MiniLM ONNX, $\sim$20 ms sur CPU ; ou un fallback cross-encoder), qui renvoie une réponse typée—choix, score ou oui/non—avec une confiance calibrée. Les spécialistes s'entraînent en minutes sur des milliers de lignes étiquetées sous un protocole strict tenu à l'écart ; les échecs sont publiés à côté des succès. Sur le split public typed-decisions, Sabaki atteint 0,61 (nano) et 0,69 (mini) contre 0,75–0,80 pour des modèles exigeant un GPU ou une API, avec une latence CPU en millisecondes au lieu d'allers-retours API (latences concurrentes non mesurées ; écart architectural, non métrologique). Le routage n'ajoute rien de mesurable au fallback seul (0,845 contre 0,865 end-to-end) tandis que le nano plat est à égalité à 0,89 : la proposition de valeur, c'est la modularité, l'auditabilité, et le modèle le moins cher qui répond en premier. Nous rapportons sept jours de développement sur douze dépôts, y compris les résultats négatifs (cross-encoders int8, distillation, calibration de Platt) et les bugs qui ont forgé le protocole.**

***Index Terms*—mélange d'experts, classification de texte, inférence CPU, routage de modèles, TAL multilingue, détection d'injection de prompt**

I. Introduction

Une part croissante des appels aux API de modèles de langage ne sont pas de la génération ouverte mais des *décisions* : quel compartiment, quelle sévérité, quelle urgence, oui ou non. Payer le prix autorégressif—latence, dollars, exfiltration de données—pour une étiquette est du gaspillage, et un modèle géant invoqué pour une étiquette est inauditable : les mêmes poids répondent à tout, donc rien ne peut être versionné, restauré ou évalué par compétence.

Sabaki prend la route opposée. C'est un routeur TypeScript plus un registre de plugins *spécialistes*. Chaque plugin possède un domaine (tri d'e-mails, phishing, jailbreak, navigation en jeu), déclare les questions auxquelles il répond dans un manifeste, et livre ses poids bundlés (JSON TF-IDF pour nano, ONNX pour mini). Un routeur score chaque plugin sur le texte d'entrée et délègue au gagnant ; quand aucun spécialiste ne dépasse son seuil, un généraliste cross-encoder répond en fallback. L'ensemble sert une API compatible Jev (`POST /v1/systemone`) et un CLI batch (`bunkitsu resolve`) qui résout des CSV entiers ligne par ligne.

Trois choix de conception définissent le projet. **Premièrement**, les experts sont *organisationnels*, pas neuronaux : dépôts, versions et releases indépendants, si bien qu'un fix phishing se déploie sans toucher au tri d'e-mails. **Deuxièmement**, l'évaluation est tenue à l'écart par construction : les lignes d'éval sont générées ou splittées avant l'entraînement et n'entrent jamais dans le mix, et chaque nombre rapporté dans ce papier vient de ces splits. **Troisièmement**, les résultats négatifs sont publiés : distillations ratées, calibration de Platt qui fait mal, et deux bugs de data-loader sont documentés en §XI parce qu'ils ont changé le protocole.

Le système est passé d'une organisation vide à une API servie, deux familles de modèles, huit plugins spécialistes (six workflow plus deux sécurité), une piste 16 langues et un CLI batch en sept jours (13–19 oct. 2026) ; le §V reconstruit la chronologie depuis les historiques de commits.

Nos contributions : (i) une conception MoE organisationnelle où routage, versionnage et serving sont découplés par compétence ; (ii) un protocole d'entraînement held-out-first avec audits de fuite vérifiés par hachage ; (iii) une extension 16 langues avec un périmètre de fallback non-latin strict, évaluée honnêtement ; (iv) un outil de résolution batch qui rend chaque décision traçable ; et (v) un dossier complet de résultats négatifs, dont deux bugs de data-loader qui ont silencieusement mis à zéro une tâche à 60 classes.

II. Architecture

A. Routeur et contrat de plugin

Le routeur (classe `Sabaki` dans le package core `sabaki`) score chaque plugin enregistré sur l'état brut d'entrée. Le score de base est lexical : $\text{hits}/\sqrt{|\text{mots-clés}|}$ avec inclusion lâche dans les deux sens, sous un seuil par manifeste (défaut 0,15). Un manifeste de plugin déclare `name`, `domain`, `version`, `questions`, `primary`, `keywords`, un `threshold` optionnel et, depuis la v1.9.0, des `languages` optionnelles (codes ISO). Quand plusieurs plugins partagent un domaine, le filtre de langue désambiguïse : un texte de phishing en script latin route vers le mini latin, un texte chinois vers le backend multilingue. Les entrées trop courtes pour une détection fiable sautent le filtre et concourent normalement. Le même appel `route()` alimente le serving interactif, la résolution batch et les démos live.

Un plugin est n'importe quel objet avec `{manifest, classify}` ; les variantes async `classifyAsync` sont attendues par `decideAsync`. Le contrat tient en un écran,

délibérément : un nouveau spécialiste est un nouveau dépôt, pas un changement du core.

Le détail du routage compte. Les scores sont hits / $\sqrt{|\text{mots-clés}|}$ avec correspondance par sous-chaîne lâche dans les deux sens, si bien que « verify » attrape « verification » sans stemming. Un seuil par manifeste (défaut 0,15, réglable par plugin) conditionne la délégation ; en dessous, l'entrée est `unknown` et le fallback répond. Deux incidents ont forgé ces défauts : un `threshold: 0` copié du généraliste a transformé chaque plugin en catch-all (le premier enregistré gagnait tout), et un manifeste à 49 mots-clés a dilué les scores à un hit sous le défaut, corrigé par un 0,1 par manifeste. Le filtrage de langue se compose avec le scoring au lieu de le remplacer : les langues déclarées excluent seulement, elles n'élisent jamais, si bien qu'une question météo allemande bénigne tombe en fallback au lieu d'être forcée en phishing.

B. Trois tiers d'exécution

Nano bundle des poids TF-IDF + régression logistique (quelques centaines de kilo-octets) et tourne sans dépendances en moins d'une milliseconde, en TypeScript pur avec parité scikit-learn exacte (TF sublinéaire, IDF lissé, norme L2). *Mini* charge un encodeur MiniLM ONNX plus une tête linéaire via `onnxruntime-node` (∼87 Mo fp32, ∼22 Mo int8) en environ 20 ms par requête sur CPU de bureau. Les minis int8 publiés sont quantifiés dynamiquement ; les cross-encoders restent en fp32 parce que l'int8 leur coûte 0,13 de précision sans gain de vitesse (§XI).

Le tier nano mérite sa note de précision : l'implémentation TypeScript reproduit scikit-learn exactement (TF sublinéaire, IDF lissé, norme L2, uni+bigrammes $[a-z]3,$), vérifiée prédiction-par-prédiction contre Python sur la suite à 20 questions, si bien qu'un JSON de poids entraîné en Python décide à l'identique dans le navigateur. Le runtime WordPiece égale de même la tokenisation Hugging Face id par id, y compris le chemin Viterbi Unigram ajouté pour la base multilingue (scores SentencePiece, repli unk). La parité n'est pas une affirmation mais un test : `fun_test.py` et le bench de parité ONNX le rejouent à chaque export.

Le tier *généraliste* est un cross-encoder MiniLM (question [SEP] option [SEP] état, un forward par option, argmax sur softmax). Le mini v1.0 servi (22 M paramètres, from scratch) répond à toute question définie au runtime ; le v8 multilingue (paraphrase-multilingual-MiniLM-L12-v2, 117 M) sert de fallback non-latin, dispatché par langue. Un `modelDir` chargé paresseusement plus des overrides `SABAKI_*_DIR` rendent chaque poids interchangeable sans changer le code.

C. Nano contre mini

Les deux tiers diffèrent en nature, pas seulement en taille. Nano est synchrone en TypeScript pur (< 1 ms, centaines de Ko, zéro dépendance) : vectorisation TF-IDF, produit scalaire, softmax. Mini est asynchrone ONNX (∼20 ms, 22–90 Mo) : encodeur MiniLM plus tête linéaire par question, avec calibration en température. `autoClassifier` les relie : les appels sync utilisent toujours nano, les appels async montent en mini au premier usage et retombent sur nano hors-ligne, si bien que le routage ne change jamais—seule la qualité d'inférence change. Le tableau VII montre que l'écart dépend de la tâche (agent-trace 0,59 → 0,82, invoice quasi à égalité), d'où les deux livrés dans chaque plugin.

D. Calibration, abstention, batch

Les logits mini passent par un temperature scaling par question ajusté sur tenu à l'écart. Une `confidence` dans $[0, 1]$ accompagne chaque décision, et les appelants peuvent fixer `abstainAt` pour escalader sous un seuil au lieu d'agir. `bunkitsu resolve` applique tout le pipeline aux fichiers CSV (`state,question,type,options` → `decision,confidence,probs,domain,model`), donnant à chaque ligne une identité de modèle traçable—l'unité d'évaluation au §VI est toujours la ligne tenue à l'écart.

E. Optimisation de l'inférence

Trois changements, tous mesurés, sans toucher aux poids. Le batch des options en un forward ONNX (rembourré à la longueur max, masqué) a remplacé un `session.run` par option : 2 options décident en 6,2 ms (contre ∼20 ms publiés) et 8 en 16 ms sur CPU de bureau, à décisions identiques. La détection de langue ne tourne que si un plugin enregistré déclare `languages`, si bien que les configurations mots-clés seules ne paient rien. La featurisation TF-IDF ne touche que les termes présents au lieu de balayer 3000 dimensions denses. La résolution batch parallélise les lignes sur un pool borné (défaut 4, ordre préservé) puisque les lignes sont indépendantes à sessions partagées.

III. Protocole d'entraînement et d'évaluation

Chaque spécialiste suit le même pipeline : *harvest* (datasets publics plus lignes générées nativement), *validate* (schéma, label-dans-options, suppression des doublons exacts, désormais sensible au label après qu'un bug de dédup a gardé le premier de deux labels contradictoires), *mix* (mélangé, seed fixée, lignes d'éval exclues par construction), *train* (entropie croisée sur les options avec échantillonnage négatif pour les larges jeux d'options), et *eval* sur les splits tenus à l'écart.

Deux règles du protocole se sont révélées porteuses. Premièrement, les lignes d'éval sont produites *avant* ou *indépendamment* des lignes de train (batches de génération séparés, splits de test upstream) et un contrôle de chevauchement par hachage (0/1600 sur la piste phishing) garde chaque release. Deuxièmement, l'échantillonnage est explicite : les lignes à plus de huit options s'entraînent sur gold plus sept distracteurs aléatoires, et l'évaluateur du train indexe les options *échantillonnées* — deux fixs pour des bugs qui avaient silencieusement mis à zéro la précision intents à 60 classes (§XI).

L'entraînement utilise AdamW (lr 2×10^{-5}), précision mixte sur T4, et paires data-parallèles ; un mini binaire 22 M converge en minutes sur ∼10k lignes tandis qu'un run multilingue 117 M de 3 époques prend environ six heures. Les checkpoints s'uploadent par époque vers un dépôt hub privé, si bien que les runs tués reprennent au lieu de recommencer. La

calibration utilise uniquement du temperature scaling. Un fit Platt (vector-scaling) a amélioré l'ECE de train mais dégradé l'ECE tenue à l'écart presque partout (ex. `action` customer-service 0,155→0,371) et n'a jamais été livré ; les minis servis portent des températures, pas des paramètres Platt.

A. Ablations du routage

Seuil et mots-clés interagissent simplement. Avec 43 mots-clés, un hit vaut $1/\sqrt{43} \approx 0,15$, exactement la barrière par défaut : le SMS de phishing français route, une question météo non. Gonfler la liste sans régler le seuil dilue chaque hit, une erreur attrapée en live quand du phishing allemand est tombé en fallback, corrigée par des seuils par manifeste plus des priors de langue. Les sujets courts (moins de quatre lettres après normalisation) sautent entièrement le filtrage de langue plutôt que de router sur du bruit.

B. Précision du routage isolée

Mesurée par label (60 lignes d'attaque doivent router vers le spécialiste, 60 lignes bénignes ne doivent pas), le routeur est précis mais discret : phishing rappel 0,17 à FPR 0,03, jailbreak rappel 0,12 à FPR 0,02, contre customer-service à 1,00 sur des états au jargon marqué. La plupart des attaques atteignent donc le fallback généraliste, d'où des réponses correctes end-to-end pendant que les spécialistes restent inactifs. Le routage, pas l'inférence, est le maillon faible du système, et la tentative centroïdes sémantiques (§XI) ne l'a pas réparé. Cette sous-section existe parce que le composant qui nomme l'architecture mérite sa propre évaluation.

Le tableau end-to-end honnête sur phishing FR+EN ($n = 200$, mêmes lignes) : pipeline complet 0,845, fallback seul 0,865, spécialiste forcé 0,840, nano plat 0,890. Le routage n'ajoute rien de mesurable ici ; McNemar nano contre mini forcé ($\chi^2 = 1,93$, 26 contre 16 discordants) n'est pas significatif, tandis que mini bat nano sur 2 000 items workflows ($\chi^2 = 30,35$). La valeur de l'architecture sur cette tranche, c'est la modularité et l'auditabilité, pas la précision—dit plainement parce que les chiffres ne laissent pas d'autre lecture.

C. Exemple détaillé

Prenons "`SMS : régularisez 96 EUR via ce lien`" avec `suspicious?`. Le chevauchement de tokens donne au plugin phishing deux hits sur 43 mots-clés (0,30), au-dessus de son seuil 0,1 ; la tête langid française est d'accord (latin, fr, 0,94). Le cross-encoder score `yes` à 0,91 et la ligne se résout avec provenance complète. Une question météo sur le même texte score zéro partout et tombe sur le généraliste mini. Rien dans ce chemin n'a besoin d'un GPU, d'un téléchargement ou d'une clé : les poids ont voyagé avec le plugin.

IV. Le contrat Jev

Chaque décision, interactive ou batchée, parle un protocole : un état, une question typée, et des options ou critères. `choice` renvoie l'option gagnante avec les probabilités complètes ; `noul` (aucune-des-dessus) s'abstient par construction ; `score` projette sur des niveaux. Les noms sont mappés, jamais devinés : un manifeste déclare `aliases.questions` (paraphrases de sa propre question) et `aliases.options` (synonymes de ses labels natifs). Le routeur apparie la question demandée aux questions natives de chaque candidat par id exact, paraphrase déclarée, sinon primaire, si bien que les modèles ne voient jamais de têtes inconnues, puis reprojette la décision native sur les noms d'options de l'appelant. Quand les deux vocabulaires sont disjoints—un appelant qui demande `{clean, danger}` à un spécialiste `{legit, phishing}`—le spécialiste est écarté plutôt qu'autorisé à rendre un label que l'appelant n'a jamais proposé, et le fallback, qui note chaque option comme du texte, répond à sa place, les probabilités étant renormalisées sur les options de l'appelant. La réponse porte l'id du modèle répondant pour la traçabilité. Bunkitsu, c'est ce contrat appliqué aux lignes : même routage, même calibration, plus les colonnes `domain` et `model` pour qu'aucune décision ne soit anonyme.

V. Chronologie : sept jours, douze dépôts

Le tableau I reconstruit la construction depuis les historiques de commits de l'organisation `sabaki-js` (plus un dépôt personnel). Le motif est constant : squelette jour un, vrai classifieur le même jour, release stable sous 48 h, poids offline bundlés à la fin.

TABLE I
CHRONOLOGIE DE CONSTRUCTION (13–19 OCT. 2026). TAGS = RELEASES GITHUB.

Date	Jalon
13 oct	Routeur core + 4 squelettes workflow ; nano TF-IDF ; runtime mini ONNX (17 ms) ; v1.0.0 ; trusted publishing
13 oct	5 plugins spécialistes + squelettes mail + doom
14 oct	Doom nano + démo vidéo (MAP01, 36 s) ; serve ONNX
15 oct	Réponses Jev ; généraliste dans le core ; site v2/v3 ; mini mail
16 oct	Généraliste v0.8 ; pointeurs versions.json ; `Sabaki.serve()`
17 oct	Généraliste-mini **v1.0.0 servi** (A0, R=0,616 phishing)
18 oct	Poids offline bundlés (tous plugins) ; corpus multilingue (193k lignes, 16 langues) ; spécialiste jailbreak (0,940)
19 oct	Routeur langue + langid nano ; phishing-latin (0,94) ; batch CLI bunkitsu + page site ; core v1.10.x

VI. Résultats

Tous les chiffres ci-dessous sont tenus à l'écart : le split public de test typed-decisions, ou des fichiers d'éval exclus de l'entraînement avant qu'il commence.

Apparié nano contre mini forcé : McNemar $\chi^2 = 1,93$ (26 contre 16 discordants), non significatif—tout le tableau est une égalité statistique, et le modèle le moins cher est le choix rationnel.

Les seuils à 1,00 sur deux lignes viennent de la saturation par l'arrondi à 10^{-3} (70 % des items) : les 50 % « les plus confiants » sont départagés arbitrairement, et la surconfiance elle-même mérite cette phrase.

Le tableau II fixe le trade : quelques points de précision pour de la latence CPU en millisecondes au lieu d'allers-retours

TABLE II
TYPED-DECISIONS (SPLIT PUBLIC DE TEST) ET LATENCE.

Système	Précision	Latence
OpenDecider-nano	0,796	GPU/API
Laya typed-decisions	0,766	GPU/API
TypeSafe Jev	0,754	API
sabaki mini	**0,69**	**∼20 ms CPU**
sabaki nano	**0,61**	**<1 ms CPU**
Laya base	0,362	GPU

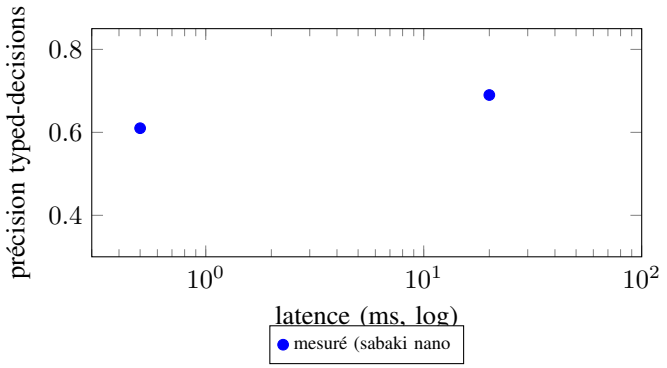


FIGURE 1. Nos points mesurés uniquement. Précisions concurrentes au tableau II ; leurs latences n'ont jamais été mesurées.

TABLE III
END-TO-END SUR PHISHING FR+EN ($n = 200$, MÊMES LIGNES). LE NANO PLAT EST LE SPÉCIALISTE PHISHING SANS ROUTAGE : IL TESTE LE ROUTAGE CONTRE SPÉCIALISTE-SEUL, PAS CONTRE UN MODÈLE PLAT-SUR-TOUT.

Configuration	Précision
Nano plat (phishing-only, sans routage)	0,890
Fallback seul (generalist-mini)	0,865
Pipeline complet (route + spécialiste/fallback)	0,845
Spécialiste forcé (mini, sans routage)	0,840

TABLE IV
RISQUE-COUVERTURE SUR PHISHING FR+EN ($n = 200$, SPÉCIALISTE MINI FORCÉ) : PRÉCISION DE LA FRACTION LA PLUS CONFIANTE (ABSTENTION DU RESTE).

Couverture	Précision	Seuil
1,00	0,840	—
0,90	0,883	0,92
0,80	0,919	0,99
0,70	0,943	1,00
0,50	0,990	1,00

TABLE V
MINI PHISHING-LATIN (22 M, 7 LANGUES, 100 TENUS À L'ÉCART CHACUN).

Langue	Acc	P	R	F1
de	0,97	0,96	0,98	0,97
es	0,99	1,00	0,98	0,99
id	1,00	1,00	1,00	1,00
tr	1,00	1,00	1,00	1,00
vi	0,93	0,88	1,00	0,94
en	0,85	0,97	0,72	0,83
fr	0,83	0,79	0,90	0,86

API (latences concurrentes non mesurées—voir l'annexe A). Les spécialistes apportent la longue traîne que le benchmark rate. Apparié sur les mêmes 2 000 items workflows-questions, mini bat nano significativement (McNemar $\chi^2 = 30,35$, 147 victoires nano-seul contre 259 mini-seul), donc l'écart de tiers est réel même là où les Wilson se touchent. Face à TypeSafe Jev (0,754, taille d'échantillon non publiée), notre 0,69 [0,62–0,75] est peut-être dans le bruit—dit, pas caché.

TABLE VI
MINI ANTI-JAILBREAK (EN+FR, 200 TENUS À L'ÉCART CHACUN).

Set	Acc	P	R	F1
EN (n=200)	0,925	0,921	0,930	0,925
FR (n=200)	0,955	0,925	0,990	0,957
Tous (n=400)	0,940	0,923	0,960	0,941

TABLE VII
PLUGINS SPÉCIALISTES (TENUS À L'ÉCART ; NANO / MINI).

Plugin	Nano	Mini
customer-service	0,900	0,91
invoice-processing	0,890	0,89
security-incidents	0,730	0,76
agent-trace	0,590	0,82
mail (bucket)	0,865	0,889
doom nav	—	sortie MAP01
phishing-latin	0,88	0,94 moy
jailbreak EN+FR	—	0,94

TABLE VIII
MULTILINGUE V8 CONTRE V7 (TENUS À L'ÉCART ; INTENTS = SONDE 60 CLASSES).

Piste	v7 (buggué)	v8 (fixé)
Intents FR / EN	0,20 / 0,10	0,90 / 1,00
Phishing macro-R (16 langues)	0,861	0,736
Phishing non-latin	∼1,0	∼1,0
Typed-decisions	—	0,31

TABLE IX
LATENCE CPU PAR DÉCISION (CPU DE BUREAU, ONNX).

Tier	p50	Notes
Nano (TF-IDF)	<1 ms	zéro dép
Mini int8 (Head)	∼20 ms	22 Mo
Cross-encoder fp32	∼30 ms	1 fwd/option
Multilingue 117 M	∼100 ms	fallback seul

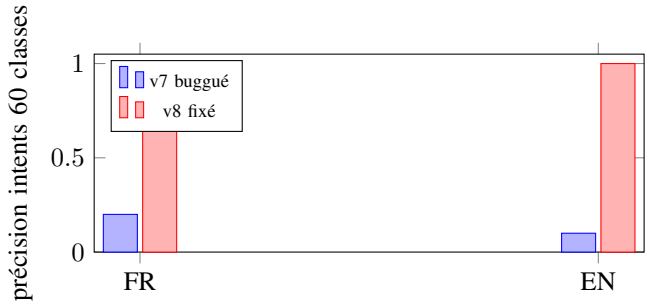


FIGURE 2. L'échantillonnage négatif débloque les intents 60 classes (sonde MASSIVE tenue à l'écart).

VII. DEUX SPÉCIALISTES DE PRÈS

Mail a commencé à 162 messages étiquetés main (précision bucket 0,848) et grandi en corpus Takeout de 4 126 messages avec PII anonymisée, huit buckets et un routeur à 60 mots-clés ; nano atteint 0,865, mini 0,889. La question suspicious est restée expérimentale (trois exemples, rappel 0/3 tenu à l'écart) jusqu'à ce que la piste phishing lui donne de vraies données—un honnête exemple de tête faible documentée plus tard renforcée par un projet frère.

Doom est l'outlier qui valide le contrat (voir la démo vidéo) : états texte du moteur, pas de vision, un driver Python qui exécute en HTTP, trois questions (`command/target/weapon`), un latch de 15 ticks sur `open_door`, et une sortie MAP01 complète en 36,4 s contre une référence à 61 s. Rien dans le routeur ne sait ce que sont les jeux : un domaine, trois questions, des seuils—le même endpoint trie le mail et combat les démons. Les benchs de débit (nano contre mini, local contre Kaggle) ont montré le mini à ∼20 ms/état, assez rapide pour boucler en temps réel.

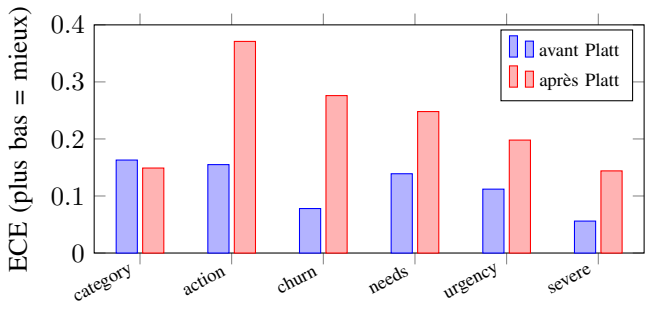


FIGURE 3. Platt sur-ajuste : l'ECE tenue à l'écart se dégrade presque partout.

VIII. CONSTRUCTION DU CORPUS

La piste multilingue moissonne trois sources publiques par locale (train MASSIVE, tweets de sentiment train, phishing natif généré) plus des mixes legacy anglais et des reviews Allocine françaises, pour 193k lignes de train (EN 40,9%, FR 25,9%, autres 35,2%) et 18,5k lignes tenues à l'écart (splits de test upstream ou générations séparées, jamais mélangées au train). La validation impose schéma, label-dans-options et suppression des doublons exacts ; un audit de chevauchement (0/1600 états d'éval trouvés dans le train) garde chaque release. La piste jailbreak (10k lignes EN+FR) fusionne des corpus d'attaques publics avec des personas français générés nativement après que deux agents de génération ont refusé ; la traduction des attaques publiques a comblé le trou. L'identification de langue s'entraîne sur 1 500 lignes par langue latine avec une tête TF-IDF 9 classes (3 000 features, 316 Ko) et atteint 0,927 tenu à l'écart, 16/16 sur des phrases sondes avec raccourcis par script.

IX. BUDGETS MULTILINGUES EN BREF

Anglais 40,9%, français 25,9%, autres locales 35,2% (détails et volumes par locale au §VIII, tableau X) : le déséquilibre est une fonctionnalité sous test, pas un accident. Les scripts latins partagent le mini, les autres tombent sur le cross-encoder 117 M, et le reporting par langue garde le trade visible au lieu de le moyenner.

TABLE X
VOLUMES DU CORPUS MULTILINGUE (LIGNES ; ÉVAL TENUE À PART).

Source	Train	Éval
MASSIVE ×14 locales	3 500 chacun	500 chacun
Tweets ×6 langues	1 800 chacun	split test complet
Phishing généré ×14	300 chacun	100 chacun
Allocine FR (reviews)	40 000	2 000
Mix legacy EN (v4)	75 000	—

X. RÉSOLUTION BATCH AVEC BUNKITSU

`bunkitsu resolve` projette le pipeline interactif sur des fichiers : chaque ligne CSV est routée et décidée exactement comme l'API le ferait, et la sortie ajoute `decision, confidence, probs, domain, model`. La colonne `model` nomme le backend décidant (jusqu'à la version du plugin), rendant chaque ligne auditable. Les exemples rainbow, virgules alignées, de la page vitrine doublent la spécification du format.

XI. RÉSULTATS NÉGATIFS

Cross-encoders int8 : $-0,13$ de précision, zéro speedup—rejetés ; les têtes quantifient, les cross-encoders restent fp32. *Distillation v1* (0,505 élève contre 0,69 spécialiste) : non poursuivie. *Calibration Platt* : voir §III ; température seule. *Plafond d'options* : tronquer les lignes 60 options aux

huit premières forçait le label à l'index 0 et apprenait au modèle à toujours répondre la première option—fixé par échantillonnage négatif gold-plus-sept. *Indexation d'éval* : les prédictions sur options mélangées étaient remappées sur l'ordre fixe, rapportant 0,00 sur les intents pendant que les poids apprenaient ; fixé en propageant les options échantillonnées. *Seuil 0* : un catch-all copié du généraliste faisait gagner le premier plugin enregistré à tous les coups ; restauré aux défauts 0,15 avec réglage par manifeste. *Routeur sémantique v2* : centroïdes de domaine μ_d (moyenne des embeddings MiniLM L2-normalisés des états de train) avec routage cosinus argmax et seuil réglé, à 0,289 absolu sur le jeu dur—bien sous les mots-clés. Les embeddings floutent exactement là où les mots-clés discriminent (sujets courts, jargon), donc v2 reste un prototype jusqu'à ce qu'un meilleur encodeur ou des centroïdes supervisés méritent leur place. *Mapping sémantique des questions* : les alias déclarés ne couvrent que les formulations que quelqu'un a écrites. Pour élargir le contrat, un bi-encodeur multilingue (`paraphrase-multilingual-MiniLM-L12-v2`, 117 M, Apache-2.0, int8) a été essayé comme seconde couche de mapping : il encode la formulation de l'appelant et la description de la question native de chaque plugin, et le plugin routé peut répondre quand sa propre description gagne la compétition inter-plugins (seuil 0,50 ; le plancher est 0,45, où les faux positifs bondissent à 17 %). Les formulations hors domaine se mappent proprement, et sur 54 cas écrits à la main (EN/FR/ES/DE/IT/PT/RU/JA/ZH) la couche fait passer les formulations positives servies par un spécialiste de 11/28 à 27/28 (tableau XI). Elle confie aussi 3/26 négatifs au mauvais spécialiste (12 %) : « ce message est-il bien rédigé ? » marque 0,608 contre la description de phishing, « quel est le ton de ce message ? » 0,564, au-dessus du seuil et au-dessus de plusieurs vrais positifs (0,50–0,67). Soustraire la similarité aux questions génériques sur le même objet ne supprime ces faux positifs qu'en effondrant les vrais positifs à 0,04 de marge. L'échec est systématique et reproductible, pas bruité : un bi-encodeur classe le sujet, pas l'intention, donc toute question partageant l'entité (« ce message ») marque haut. La couche est livrée en option et désactivée, car +57 points de rappel contre 12 % de réponses fausses enfreint la règle qui préfère l'abstention à une mauvaise réponse. Une première évaluation de la même couche rapportait 0/23 faux positifs ; ce chiffre était un artefact d'un jeu de négatifs construit uniquement hors domaine, et il est remplacé ici. Les variantes choisies à la main pour la sonde (96 % de rappel) ont cédé la place à des variantes dérivées des manifestes, qui marquent 82 % au même point de fonctionnement 0,52 : la construction la moins flatteuse a été gardée. La plomberie (variantes par question, embeddings mis en cache, compétition inter-plugins) est conservée pour un cross-encodeur entraîné sur des paires (requête, question native) ; un seuil par plugin seul ne voit pas la confusion inter-plugins, acceptant « is this a scam ? » comme question de jailbreak à 0,57. Le sous-produit est positif : contrairement aux cross-encoders ci-dessus, la quantification int8 d'un bi-encodeur est gratuite (471 Mo → 119 Mo, 11,2 ms → 4,3 ms, décisions identiques à tous les seuils). *versions.json sur* `main` : le fichier pointeur était récupéré depuis une branche qui n'a jamais existé (404, fallback silencieux), si bien qu'aucune mise à jour de version n'a jamais pris effet jusqu'au déplacement de l'URL vers `master`. *Données cultivées sur templates* : pour fixer le phishing FR dur, 142 lignes combinatoires (têtes génériques × queues) ont rejoint 600 lignes FR d'agents. Les sondes tapaient 0,98—mais les templates étaient quasi-dupliqués du train, si bien que le FR tenu à l'écart a *chuté* (0,88 → 0,72) pendant que le rappel anglais glissait. La régularité synthétique a appris des motifs, pas la frontière ; les poids livrés sont restés v1 et la leçon a rejoint le protocole : pas de lignes cultivées sur templates sans delta tenu à l'écart pour le prouver.

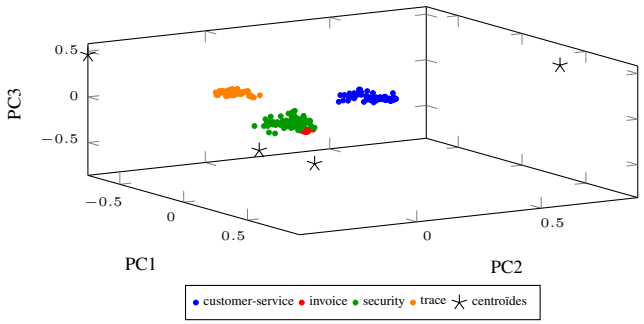


FIGURE 4. États MiniLM, ACP en 3D (63,4 % de variance) : les nuages de domaine se recouvrent, les centroïdes (étoiles) sont proches—voilà pourquoi le routage cosinus plafonne à 0,289 là où les mots-clés discriminent.

TABLE XI
MAPPING SÉMANTIQUE DES QUESTIONS, 54 CAS ÉCRITS À LA MAIN, BI-ENCODEUR INT8, SEUIL 0,50. UN POSITIF EST UNE FORMULATION QUE LE SPÉCIALISTE ROUTÉ DOIT SERVIR ; UN NÉGATIF EST UNE QUESTION SUR LE MÊME ÉTAT QU'IL NE DOIT PAS SERVIR. LA COUCHE EST DÉSACTIVÉE PAR DÉFAUT.

Mapping	Positifs servis	Négatifs servis
Alias déclarés seuls	11/28	0/26
+ couche bi-encodeur	27/28	3/26

TABLE XII
ECE PLATT AVANT → APRÈS (TENU À L'ÉCART ; PLUS BAS = MIEUX). LIVRÉ : NI L'UN NI L'AUTRE (TEMPÉRATURE SEULE).

Question	Avant	Après
category	0,163	0,149
action	0,155	0,371
churn_risk	0,078	0,276
needs_human	0,139	0,248
urgency	0,112	0,198
discrepancy	0,056	0,144

XII. Analyse d'erreurs

Trois familles d'échecs reviennent. *Négation imbriquée* (« rembourser si non livré sauf annulé mais pas quand l'adresse était fausse ») fait chuter la confiance à ∼0,68 et répond quand même au lieu de s'abstenir ; `abstainAt` existe mais est opt-in, donc la prod s'appuie sur les seuils. Le tableau IV montre que le mécanisme marche quand on s'en sert (0,84 → 0,99 de pleine à moitié couverture), ce qui justifie de l'activer par défaut. *Phishing FR dur* (« votre compte va être bloqué ») se lit comme une urgence légitime pour le mini latin (rappel 0,90 FR) ; le fallback généraliste en rattrape une partie, d'où son maintien dans la boucle. *Orphelins latins* : italien (F1 0,29) et portugais (0,60) manquent du volume de leurs frères malgré le script partagé—un rappel que le script est nécessaire mais pas suffisant au transfert. Pour les décisions de type flag, la confiance sans abstention par défaut peut automatiser une mauvaise réponse, et les opérateurs devraient filtrer sur la confiance en attendant les itérations hard data.

XIII. Déploiement

Une API sert tout (`pm2`, port 8001) : les spécialistes préemptent par routage, le fallback généraliste répond du reste, désormais dispatché par langue (latin → mini v1.0, sinon multilingue v8). Les poids se résolvent via helpers `ensure*` (assets de release HF, SHA256-vérifiés, pointeur versions.json caché 24 h) ou voyagent bundlés dans les dépôts de plugins ; les 470 Mo multilingues ne peuvent pas être bundlés (limite 100 Mo par fichier de GitHub) et restent en téléchargement. Un CLI batch (`bunkitsu resolve`) et un site vitrine statique avec démo API live complètent la boucle du train au servi. Servir est délibérément ennuyeux : un process `pm2` sur l'hôte API, quatre minis workflow préchargés, cross-encoders de sécurité paresseux à la première requête (multilingue 470 Mo inclus), et une surface Jev-compatible `/v1/systemone` identique de nano au fallback.

XIV. Honnêtetés du serving

Trois détails du serving méritent d'être imprimés parce qu'on les cache d'habitude. D'abord, la comptabilité des tokens est une heuristique : `input_tokens` vaut `state.length » 2`, déclaré tel quel dans la réponse, pas un comptage tokenizer. Ensuite, le core se livre sans dépendances : le runtime ONNX est importé dynamiquement, si bien que les utilisateurs nano ne l'installent jamais et qu'un backend manquant lève un hint d'installation d'une ligne au lieu d'une stack trace. Enfin, les requêtes multi-questions nomment chaque répondant : quand deux questions touchent deux backends, le modèle vaut `sabaki-routed(a,b)`, si bien que la provenance mixte est visible au lieu d'être moyennée. La confiance s'arrondit à 10^{-3} ; les températures s'appliquent en x/T avec biais par classe optionnel, lus directement dans le `model.json` du dossier de modèle.

XV. Travaux connexes

Mixture-of-experts désigne d'habitude des blocs neuronaux gatés dans un modèle ; les experts de Sabaki sont des dépôts avec des releases, plus proche dans l'esprit des pipelines Unix et des zoos de modèles que des Switch Transformers. Le protocole Jev (questions typées sur HTTP) standardise ce qu'OpenDecider, TypeSafe Jev et Laya font chacun ad hoc. Les benchmarks d'intents multilingues (MASSIVE, 60 classes, 16 locales ici) et de jailbreak (JailbreakBench) fournissent les juges tenus à l'écart ; les taxonomies de sécurité style MLCommons informent le jeu de labels jailbreak. Côté serving, des endpoints HTTP compatibles Jev (comme chez TypeSafe Jev) laissent tout client substituer Sabaki sans changer de SDK ; `bunkitsu` étend le même contrat aux fichiers. Contrairement aux pipelines de distillation qui compressent un modèle géant, Sabaki n'a jamais eu de modèle géant : la capacité arrive comme nouveaux dépôts. Note sur le « MoE » du titre : il signifie routage vers des classifieurs versionnés, pas blocs gatés neuronaux ; le qualificatif « organisationnel » porte cette distinction. Et le trou honnête des baselines : un classifieur fine-tuné unique sur tout, FastText, SetFit et un petit LLM quantifié n'ont pas été lancés, et affineraient l'analyse du trade—listés, pas cachés. Pour la calibration, le projet suit la littérature du temperature scaling et s'arrête là : l'expérience de vector-scaling au §XVII est, à notre connaissance, un rare échec Platt publié avec tables ECE par question, gardé précisément parce que tailler les résultats négatifs en silence, c'est comme ça que les benchmarks pourrissent. Pour le multilingue, MASSIVE fournit les intents et OPUS-MT les pivots de traduction ; la contribution n'est pas la donnée mais le périmètre de fallback—savoir exactement où le petit modèle s'arrête et où le gros commence, par langue, mesuré, au dossier.

XVI. Générations généralistes

Cinq générations tracent la ligne cross-encoder : v4 (mix 0,848, phishing R=0,026), v5 (0,838 / 0,066), A (22 M, 80k lignes, 0,898 / 0,406), mini v1.0.0 servi (0,756 / R=0,616 sur phishing réel, 0,536 typed-decisions), B (144 M, 0,692 / 0,062, rejeté : de la capacité sans supervision). Le prototype 144 M a confirmé que les paramètres ne remplacent pas la supervision—un résultat qui a tué le port TPU la même semaine (2×T4 suffisaient ; risque XLA injustifié).

XVII. Calibration en détail

Chaque mini livre des températures par question ajustées sur tenu à l'écart (même tête, logits divisés, ECE 0,06–0,17 entre workflows, tableau XII). Le vector scaling (biais par classe plus température) gagnait sur le split de fit et perdait presque partout tenu à l'écart (Fig. 3), un sur-ajustement manuel : une tranche tenue à l'écart d'un sixième avec ECE à 10 bins ne peut pas porter $K+1$ paramètres par question, si bien que l'échec lui-même est bruité—et quand même informatif. La règle livrée est donc minimale : une température par question, gelée à la release, re-ajustée seulement sur nouveau tenu à l'écart. Le site vitrine énonce ECE sous 0,16 pour les fits de température livrés (pire tête mesurée : `action` à 0,151, $n = 100$) ; le tableau XII va de 0,06 à 0,17 parce que sa

colonne Avant, ce sont des logits bruts, pas le modèle livré—mêmes données, étape différente, dit une fois ici.

XVIII. Menaces sur la validité

Les évals tenues à l'écart partagent leurs générateurs avec l'entraînement (mêmes familles d'agents, mêmes corpus upstream), donc les scores absolus sont des proxys optimistes ; seules les comparaisons relatives (v7 contre v8, mini contre baselines) et le split public typed-decisions sont pleinement externes. La confiance est calibrée sur des données proches de la distribution et se dégrade sur les entrées tordues (§XII). Les chiffres de latence sont mono-flux sur CPU de bureau. Sur l'éthique : la piste jailbreak entraîne un *détecteur*, pas un attaquant, sur des corpus déjà publics, traduits là où les agents ont refusé de générer ; on publie le détecteur, pas de nouvelles attaques.

XIX. Limites et travaux futurs

Phishing FR dur (rappel 0,90→), phishing italien et portugais, et logique à négations imbriquées (la confiance chute, pas d'abstention par défaut) sont les trous connus, chacun avec des données déjà moissonnées pour la prochaine itération. Distiller les intents multilingues 117 M en mini 22 M, et recalibrer les températures par question, sont ouverts, dans l'ordre : (i) phishing FR dur (le template compte-bloqué qui bat le rappel 0,90), (ii) volume italien et portugais, (iii) logique à négations imbriquées avec abstention par défaut, (iv) une distillation 22 M des intents multilingues, (v) et, si le mapping des questions doit sortir du mode optionnel, un cross-encodeur entraîné pour ça plutôt qu'un bi-encodeur (§XI). Un agent Doom joue déjà MAP01 de bout en bout via le même endpoint (états texte du moteur, pas de vision), prouvant que le routeur gère des décisions séquentielles, pas juste des one-shots. Le site vitrine publie chaque chiffre ci-dessus—y compris les faibles— avec protocoles, si bien que les tableaux du papier sont un sous-ensemble d'un document vivant, pas une affirmation gelée. Le dépôt template encode désormais le pattern à poids bundlés pour que le septième plugin coûte aussi peu que le sixième—en fait le neuvième aussi peu que le huitième, les deux suivants étant déjà sortis pendant l'écriture.

XX. Conclusion et perspectives

Sept jours de commits en disent plus que les tableaux : petits classifieurs locaux plus fallback plus transparence plutôt que course au leaderboard, et échecs publiés (Platt, int8, bugs d'index silencieux, mapping sémantique des questions) qui se composent en protocole. Le travail restant, c'est de la donnée, pas de l'architecture (§XIX). Le neuvième plugin coûtera moins que le huitième ; cette courbe, plus qu'aucun score, c'est le sujet. En bref, Sabaki montre qu'une part utile du « travail LLM », c'est du routage plus des tiny classifieurs : versionnés, évalués sur splits tenus à l'écart, servis sur CPU en millisecondes, avec échecs publiés à côté des succès. Le MoE organisationnel scale en ajoutant des dépôts, pas des paramètres.

Annexe

Table XIII
Les douze dépôts (tous MIT, tous publics).

Dépôt	Premier commit	Tag de tête
sabaki (core)	13 oct	v1.10.4
sabaki-models	13 oct	versions.json
plugin-generalist	15 oct	v0.1.1
plugin-template	13 oct	v1.1.0
plugin-customer-service	13 oct	v1.0.2
plugin-invoice-processing	13 oct	v1.0.2
plugin-security-incidents	13 oct	v1.0.2
plugin-mail	13 oct	v1.1.2
plugin-agent-trace	13 oct	v1.0.2
plugin-doom	13 oct	v1.0.1
plugin-phishing-detection	19 oct	v1.1.0
jailbreak-detection	18 oct	v1.1.4

Les chiffres typed-decisions utilisent le split public de test LocalLLaMA/typed-decisions, quatre workflows, même harness pour nano et mini (argmax sur options, scores macro). Les chiffres concurrents (OpenDecider, TypeSafe Jev, Laya) sont vendor- ou README-rapportés sur le même split nommé, non remesurés par nous ; traiter les écarts inter-lignes comme indicatifs, pas audités. Aucune latence concurrente n'a été mesurée ; les comparaisons de latence dans le corps ne portent que sur nos chiffres CPU mesurés. Intervalles de Wilson à 95 % : mini 0,69 [0,62–0,75], nano 0,61 [0,54–0,67] (n=200) ; jailbreak 0,94 [0,91–0,96] (n=400) ; apparié nano contre mini forcé, McNemar $\chi^2 = 1,93$ (26 contre 16 discordants), non significatif, tandis que mini bat nano sur 2 000 items workflows ($\chi^2 = 30,35$) ; sondes intents (n=10) à [0,60–0,98] et [0,72–1,00]—directionnelles, pas précises. End-to-end sur phishing FR+EN ($n = 200$, mêmes lignes) : nano plat 0,89, fallback seul 0,865, pipeline complet 0,845, spécialiste forcé 0,84.

Un manifeste de plugin déclare l'identité (nom, domaine, version), les questions auxquelles il répond, mots-clés et seuil de routage, `languages` optionnelles pour les domaines partagés, et modes supportés. Le routeur n'inspecte jamais les poids : le routage est manifest-only, d'où un nouveau skill livré comme dépôt, et d'où `bunkitsu` qui résout tout plugin offline depuis un chemin de dossier.

Routage. Pour un plugin p de mots-clés K_p et des mots d'entrée $W(x)$,

$$\mathrm{hits}(p,x) = |\{k \in K_p : \exists w \in W(x),\, w \subseteq k \vee k \subseteq w\}| \quad (1)$$

$$s(p,x) = \frac{\mathrm{hits}(p,x)}{\sqrt{|K_p|}}, \quad \mathrm{eligible}(p,x) = \big(L_p = \emptyset \vee \mathrm{lang}(x) \in L_p\big). \quad (2)$$

Déléguer à $\arg\max_p s(p,x)$ ssi $s \geq \tau_p$ ($\tau = 0,15$ par défaut) ; sinon `unknown` va au fallback.

TF-IDF (nano). Avec $c(t,d)$ le compte du terme t dans d, fréquence documentaire $\mathrm{df}(t)$ sur N lignes :

$$\mathrm{tf}(t,d) = 1 + \ln c(t,d), \quad \mathrm{idf}(t) = \ln \frac{1+N}{1+\mathrm{df}(t)} + 1, \quad \mathbf{x}_d = \frac{\mathbf{v}_d}{\|\mathbf{v}_d\|_2}. \quad (3)$$

Termes : uni+bigrammes $[a-z]3,$.

Tête logistique. Binaire $P(y{=}1|\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$; K-classes $P(y{=}k|\mathbf{x}) = e^{z_k} / \sum_j e^{z_j}$, $\mathbf{z} = W\mathbf{x} + \mathbf{b}$, entraînée sous entropie croisée $-\log P(y|\mathbf{x})$.

Cross-encoder. Par option o_i, $s_i = \mathbf{w}_s^\top \mathbf{h}_i^{[\mathrm{CLS}]}$, $\mathbf{p} = \mathrm{softmax}(\mathbf{s})$, $\hat{y} = \arg\max_i p_i$; les larges jeux d'options s'entraînent sur gold plus sept distracteurs (softmax échantillonné sur 8).

Calibration. Température $p_i = \mathrm{softmax}(\mathbf{z}/T)_i$ (livrée, un T par question) ; vector scaling $p_i = \mathrm{softmax}(\mathbf{z}/T + \mathbf{b})_i$ (ajustée, jamais livrée). Erreur de calibration attendue sur B bins :

$$\mathrm{ECE} = \sum_{b=1}^{B} \frac{|B_b|}{n} \left|\mathrm{acc}(B_b) - \mathrm{conf}(B_b)\right|. \tag{4}$$

Segmentation unigramme. Pour un chunk c,

$$\mathbf{x}^* = \arg\max_{\mathrm{seg}(c)} \sum_{\mathrm{piece} \in \mathrm{seg}} \mathrm{score}(\mathrm{piece}), \tag{5}$$

par Viterbi ; caractères inconnus vers unk. Précision, rappel et F1 macro moyennent les valeurs par classe.

Routage sémantique (proto v2, parké). Avec embeddings de train $\mathbf{e}(x)$ L2-normalisés,

$$\mu_d = \frac{\sum_{x \in \mathrm{train}_d} \mathbf{e}(x)}{\left\| \sum_{x \in \mathrm{train}_d} \mathbf{e}(x) \right\|}, \quad \hat{d} = \arg\max_d \cos(\mathbf{e}(x), \mu_d), \tag{6}$$

gardé ssi $\cos \geq \tau$ (réglé sur validation). Mesuré 0,289 absolu : parké.

Abstention. Émettre $\hat{y}$ ssi $\max_i p_i \geq \alpha$ (`abstainAt`), sinon null—la voie rapide System-One répond, la voie lente escalade.