Do Proactive Agents Need an LLM to Decide When to Act?

Anonymous authors
Paper under double-blind review

Abstract

Proactive assistants continuously decide when to intervene and what context should support the intervention. Large language model (LLM) pipelines repeatedly interpret activity histories to make these decisions, paying an inference cost even when the assistant remains silent. We show that a small graph model can handle both decisions and improve the language agents it controls. Our key insight is that user activity has a native graph structure: events involve persistent entities whose recurrence connects interactions over time. Triggering and context selection map directly to predictions on event and entity nodes. We introduce a temporal-graph-learning (TGL) controller that learns these predictions jointly and supplies both outputs in one forward pass. The downstream language agent generates suggestions on triggered events using the activity history and scored entities. At 11.13 ms per event on a GPU server, TGL achieves the highest AUCs among nine trigger architectures and gives ~ 4–$7\times$ trigger-stage speedups over the two single-forward LLM triggers. A shared TGL model improves F1 across all 14 downstream backbones by a mean of 16.7 points. The controller also runs at 13.99 ms on a consumer laptop with a $\sim$220 MiB BF16 resident footprint, bringing effective proactive control to on-device deployment.

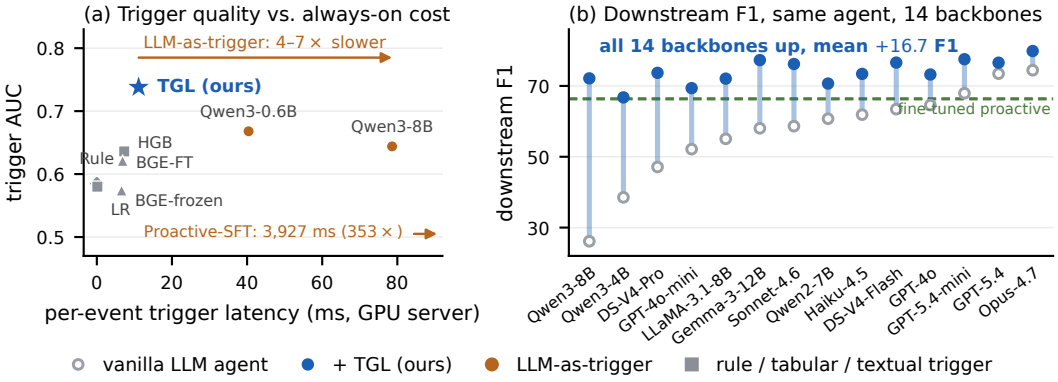


Figure 1: A lightweight graph controller supplies the always-on trigger and context-selection interface. **(a)** TGL achieves the highest trigger AUC among nine architectures at 11 ms per event; the LLM trigger protocols cost 4–353$\times$ as much (Section 5.4). **(b)** The controller improves F1 across all 14 downstream backbones by a mean of 16.7 points and exceeds both published fine-tuned proactive baselines on their corresponding frozen backbones.

1 Introduction

Proactive assistants bring large language model (LLM) agents into a user's ongoing workflow, offering help as opportunities arise across desktop and mobile activity (Lu et al., 2025; Yang et al., 2025b; 2026; Zhao et al., 2025; Tang et al., 2026). Each intervention depends on two decisions: *when* to interrupt and *what* to ground the assistance on. Recent systems use LLMs to interpret activity histories, decide whether assistance is warranted, and prepare context for action generation (Lu et al.,

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.

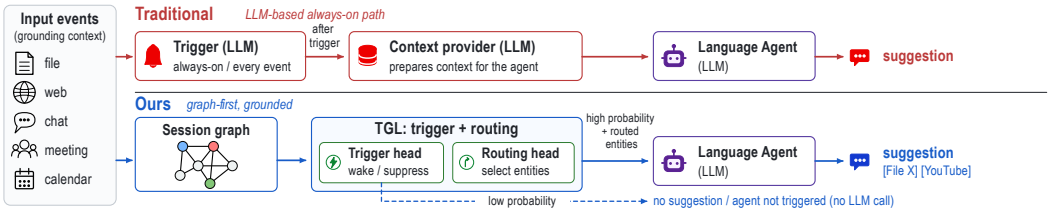


Figure 2: A shared graph controller handles triggering and context selection before language-agent generation. The upper path illustrates LLM-based implementations of these functions (Yang et al., 2025b; Xie et al., 2026). TGL computes trigger and entity-routing outputs in one forward pass and invokes the downstream language agent on triggered events.

2025; Yang et al., 2025b; Xie et al., 2026). This design places LLM inference on the continuous monitoring path and introduces two problems.

Slow and expensive triggering. LLM latency sits directly on the path to each wake-up decision. A released desktop demo defaults to a 15-second interval, or 240 checks per hour; each check processes the activity history before deciding whether to offer help (Lu et al., 2025). Even a decision to remain silent incurs this inference cost. Frequent polling trades higher resource use for faster reactions (Maldaner et al., 2026), while keeping the model available imposes a persistent memory footprint.

Indirect modeling of structured context. Existing pipelines convert structured activity records into natural-language event descriptions for LLM prediction (Lu et al., 2025), and use LLMs to condense retrieved evidence into textual context (Xie et al., 2026). This leaves event–entity association to language inference: repeated edits of the same script appear as separate mentions that the model must connect to identify the relevant artifact. Using relations already present in the activity records thus requires interpreting their textual expression at each decision. The descriptions also determine which cues reach the decision model: generic summaries can omit relevant details, while verbose descriptions introduce redundant information (Yang et al., 2025b).

Activity records associate timestamped events with entities such as files, applications, and web pages (Lu et al., 2025; Yang et al., 2026). Recurring entities connect separate events, while temporal order links successive interactions. These relations give user activity a native event–entity graph structure. Our key insight is to formulate both control decisions as node prediction on this graph: predicting whether the current event warrants assistance answers *when to intervene*, and predicting which of its entities should ground that assistance answers *what to anchor*.

We introduce a temporal-graph-learning (TGL) controller that learns these two predictions jointly. A compact graph encoder aggregates context along event–entity and temporal connections, and trigger and routing heads use this shared representation to produce a wake-up probability and entity relevance scores in one forward pass. The controller handles continuous monitoring; on triggered events, a frozen language agent receives the activity history and scored entities to generate a suggestion (Figure 2).

This simple design combines low-cost monitoring with stronger proactive performance (Figure 1). At 11.13 ms per event, TGL achieves the highest trigger AUCs among nine architectures, exceeding the best LLM trigger by 7.0 AUC points when ranking events that require assistance. A shared TGL model improves downstream F1 across all 14 backbones by a mean of 16.7 points. The activity graph thus supports a fast controller that improves the language agents it guides.

Our contributions are threefold:

- We formulate proactive control as joint prediction on event and entity nodes, and develop a shared temporal graph model that computes triggering and context selection in one forward pass.
- We show that one trained graph controller improves proactive assistance across 14 frozen language backbones with the same threshold and prompt. Ablations measure the contributions of triggering, routed context, and the downstream prompt interface.
- We demonstrate deployment at 11.13 ms per event on a server and 13.99 ms on a consumer laptop, giving $\sim$4–7$\times$ and $\sim$12–83$\times$ trigger-stage speedups over the two single-forward LLM triggers.

The controller has 1.16 M trainable parameters and a $\sim$220 MiB BF16 resident footprint including its frozen text encoder.

2 Related Work

Proactive assistance. Proactive assistance spans situation-aware agents, personal assistants, context-aware services, plan recognition, and recommendation (So & Sonenberg, 2004; Myers & Yorke-Smith, 2007; Hong et al., 2009; Oh et al., 2011; Woerndl et al., 2011). Human-computer interaction studies examine user expectations, interaction design, and satisfaction (Meurisch et al., 2020; Jones et al., 2024; Deng et al., 2024; Diebel et al., 2026). LLM-based systems extend this agenda to desktop activity (Lu et al., 2025), open-world sensory inputs (Yang et al., 2025b;a), mobile interfaces (Yang et al., 2026; Zhao et al., 2025; Kong et al., 2026), dialogue (Liu et al., 2026; Zhang et al., 2025), programming, and cooperative gameplay (Chen et al., 2025b; Zhang et al., 2024a). Recent methods use clarification and long-term memory to guide assistance (Zhang et al., 2024b; Xie et al., 2026), while reinforcement learning optimizes personalized behavior and action timing (Sun et al., 2026; Ding et al., 2026a). Lightweight controllers also gate reasoning and select candidate functions from multimodal inputs (Ding et al., 2026b), or adapt intervention timing from temporal and activity preference statistics (Wang et al., 2026). We learn triggering and entity priorities jointly on a shared activity graph.

Memory and personalization. Personalization uses user histories and preferences to adapt LLM behavior (Salemi et al., 2024; Tan et al., 2024; Richardson et al., 2023), supported by retrieval, profiles, and accumulated experience (Lewis et al., 2020; Huang & Huang, 2026; Shi et al., 2025; Yao et al., 2023; Chen et al., 2023; Zhao et al., 2024; Li et al., 2025; Wang et al., 2025). Simulated user feedback further supports learning timely, personalized recommendations (Kim et al., 2026). Temporal knowledge graphs organize agent memory (Rasmussen et al., 2025); recent egocentric assistants combine graph memory, temporal summaries, and visual retrieval to ground proactive responses (Gong et al., 2026). Evaluation connects these capabilities to the costs of interruption, attentional demand, and task resumption (Sawhney & Schmandt, 1999; Gluck et al., 2007; Iqbal & Bailey, 2008; Mark et al., 2008; Adamczyk & Bailey, 2004; Iqbal & Horvitz, 2007; Czerwinski et al., 2004; Parnin & Rugaber, 2012). Web, GUI, and mobile benchmarks provide interactive environments (Deka et al., 2017; Deng et al., 2023; Zhou et al., 2024; Gao et al., 2024; Kapoor et al., 2024; Chen et al., 2025a; Rawles et al., 2025). New proactive benchmarks assess intervention timing and content in recorded workflows (Tang et al., 2026), event tracking in ongoing conversations (Li et al., 2026), and response latency alongside resource use during extended monitoring (Maldaner et al., 2026). These tasks motivate efficient decisions over evolving activity and context.

Temporal graph learning. Temporal graph learning models evolving interactions through graph representations. Continuous-time and dynamic graph models such as TGAT and TGN encode time-dependent neighbourhoods (Xu et al., 2020; Rossi et al., 2020). Related work on dynamic knowledge and temporal reasoning further explored recurrent, evolutionary, and neighborhood-based graph updates (Trivedi et al., 2017; Jin et al., 2020; Li et al., 2021; 2022a;b). More generally, graph encoders established practical ways to model relational structure and node-local message passing (Kipf & Welling, 2017; Veličković et al., 2018; Schlichtkrull et al., 2018; Vashishth et al., 2020). Session graphs built from a user's interaction stream are likewise the substrate of session-based recommendation, where a GNN over one session's items predicts the next item (Wu et al., 2019; Xu et al., 2019; Wu et al., 2023). TGL builds on both lines as a relation-aware encoder over the heterogeneous event–entity graph. We formulate proactive triggering and context selection as joint node-level prediction: event and entity readouts determine when an agent acts and which context it receives.

3 Problem Formulation

We consider a proactive assistant that observes a timestamped event stream $\mathcal{E} = \{e_1, e_2, \ldots, e_t, \ldots\}$, where each event records a user or environment observation such as an application switch, file edit, browser visit, query, or other interaction trace. At each decision time t, the system must answer two coupled questions: *(i)* should the downstream agent be invoked now; and *(ii)* if so, which contextual entities should be routed forward as evidence?

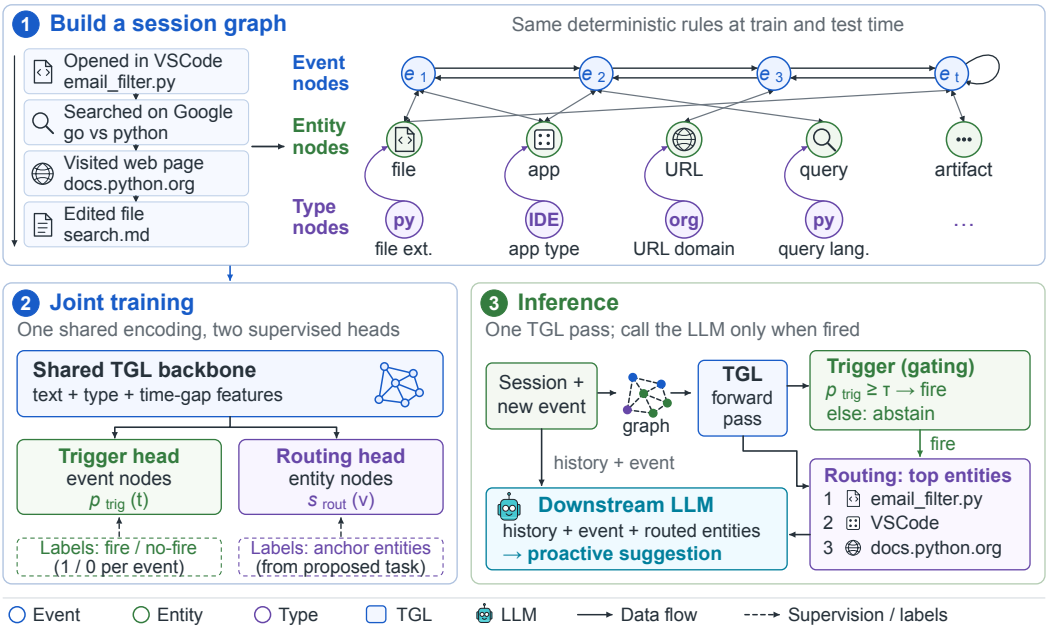


Figure 3: Joint trigger and routing prediction on the activity graph. Event, entity, and type nodes encode the session's relational structure. A shared encoder trains with event-level trigger and entity-level routing supervision. At inference, one forward pass supplies a trigger decision and scored entities to the downstream language agent.

Both decisions are produced from the observed history up to t, represented as a heterogeneous temporal graph $G_t = (V_t, E_t)$ over event nodes and the entities they touch.

Training provides binary event labels $y_t^{\text{trig}} \in \{0, 1\}$ for triggering and binary entity labels $y_v^{\text{rout}} \in \{0, 1\}$ for routing. These labels are derived from the released task suggestions by the procedure in Appendix B. A shared graph encoder supports two readouts: a trigger head on event nodes and a routing head on entity nodes. At decision time t, the trigger score determines whether to invoke the downstream agent, and the routing scores rank the entities associated with e_t. Both readouts are computed in one forward pass.

The graph controller processes each event and supplies the invocation decision and selected context. The downstream LLM receives the rolling session history and routing list, then composes the suggestion. This interface gives the two stages distinct computational roles: continuous activity prediction in the graph controller and task generation in the language agent.

4 Method

We model proactive activity as a heterogeneous temporal graph $G_t = (V_t, E_t)$ (Figure 3). Nodes represent the events themselves and the semantic entities the user touches at each event (files, applications, URLs, search queries, and lightweight *type nodes* that group entities by extension, topic, domain, or query language). Edges encode `has_entity` relations from an event to its extracted entities and the reverse `belongs_to` relation, two directional temporal edges between consecutive events (forward $t \to t{+}1$, past-informs-future; backward $t{+}1 \to t$, future-informs-past), and `self` loops. The graph is built deterministically from the event stream by a rule-based extractor, whose details (entity vocabulary, type-augmentation rules, edge channels) are given in Appendix B.

4.1 Joint Trigger and Routing as Two Heads

The graph connects the two prediction tasks through their evidence. Event nodes aggregate participating entities and nearby activity, while entity nodes aggregate the events in which they appear. Joint

trigger and routing supervision trains this shared encoder to support event-level wake-up decisions and entity-level context selection.

Node features. Every node v enters the encoder as the concatenation of a frozen text embedding of its surface string, a learned embedding of its node type τ_v, and, for event nodes, a learned embedding of the time gap Δt_v since the previous event in the session (zero for entity nodes):

$$h_v^{(0)} = W_0 \left[\text{BGE}(\text{text}_v) \,\|\, E_{\text{type}}[\tau_v] \,\|\, \text{MLP}_{\Delta t}\big(\log(1+\Delta t_v)\big) \right], \tag{1}$$

Here $\text{BGE}(\cdot)$ is a frozen text encoder and W_0 maps the features to width d. During streaming inference, recurring entities reuse their cached text embeddings; each update encodes the current event and newly encountered entities. This makes persistent entity identity useful for both context aggregation and computation reuse.

Relation-aware message passing. Information flows over the session graph through $L{=}3$ rounds of attention-weighted message passing (Brody et al., 2022), in which every edge $(u \to v)$ carries a learned embedding e_r of its relation $r \in \{\texttt{has_entity}, \texttt{belongs_to}, \texttt{fwd}, \texttt{bwd}, \texttt{self}\}$, so the attention can weight a file differently from a preceding event or from the node itself:

$$\alpha_{vu}^{(\ell)} = \underset{u \in \mathcal{N}(v)}{\text{softmax}}\Big(a_\ell^\top \,\text{LeakyReLU}\big(W_s^\ell h_v^{(\ell-1)} + W_t^\ell h_u^{(\ell-1)} + W_e^\ell \, e_{r(u,v)}\big)\Big), \tag{2}$$

$$h_v^{(\ell)} = \text{LN}\Big(h_v^{(\ell-1)} + \text{Drop}\Big(\text{GELU}\Big(\Big\|_{k=1}^{K} \sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(\ell,k)} \, W_t^{\ell,k} \, h_u^{(\ell-1)}\Big)\Big)\Big), \tag{3}$$

with K attention heads concatenated, and a residual, LayerNorm, GELU, and dropout block around each layer. $\mathcal{N}(v)$ contains the explicit `self` edge with its own relation embedding. A Jumping Knowledge readout (Xu et al., 2018) concatenates the input projection with every layer as $h_v^{\text{jk}} = [h_v^{(0)} \,\|\, h_v^{(1)} \,\|\, h_v^{(2)} \,\|\, h_v^{(3)}]$, giving both heads access to node-local features and aggregated session context.

Prediction heads and training objective. The two heads are parallel MLPs on the same h^{jk}, each applied to its own node type:

$$p_{\text{trig}}(t) = \sigma\big(\text{MLP}_{\text{trig}}(h_{e_t}^{\text{jk}})\big) \;\text{ for event nodes,} \qquad s_{\text{rout}}(v) = \sigma\big(\text{MLP}_{\text{rout}}(h_v^{\text{jk}})\big) \;\text{ for entity nodes,} \tag{4}$$

The joint objective combines trigger and routing losses:

$$\mathcal{L} = \lambda_{\text{trig}} \,\text{BCE}_{w_+}\big(p_{\text{trig}}, y^{\text{trig}}\big) + \lambda_{\text{rout}} \,\text{BCE}\big(s_{\text{rout}}, y^{\text{rout}}\big), \tag{5}$$

where w_+ down-weights the dominant positive trigger class. Both loss terms update the shared graph encoder. Hidden width, head count, loss weights, and projection sizes are in Appendix F.

At test time, the trigger fires when $p_{\text{trig}}(t) \geq 0.5$. The routing head ranks entities connected to the current event, and up to 20 entries are passed to the downstream agent on triggered events. The threshold, routed-list cap, and inference procedure are shared across all downstream backbones (Appendix F).

Anchor routing labels. The released training tasks supervise both decisions. A non-null task or an explicit acceptance annotation supplies a positive trigger label. We apply the event extractor to the task text. When its entities match the event graph, the matches supply routing positives and the remaining entities at that event supply negatives. These labels connect the wake-up target to the artifacts used in the training suggestion. Appendix B gives the complete labeling procedure for the ProactiveAgent training split, including empty-match and negative-trigger events.

Downstream agent. A frozen instruction-tuned chat model receives the rolling activity history and the scored entity list when the trigger fires. The list supplies concrete anchors, and the history supplies the surrounding task context. The language agent uses both to compose a user-facing suggestion in a JSON proposed-task field; with an empty list, it draws on the history alone. Prompt design and few-shot construction are described in Appendix C, and the same prompt serves every backbone.

5 Experiments

We evaluate two requirements of the design: a shared controller should support different language agents and remain inexpensive to run continuously. We first compare suggestion quality across backbones, then examine invocation timing and routed context through pipeline ablations. Trigger-architecture and deployment comparisons assess the quality and cost of the always-on decision.

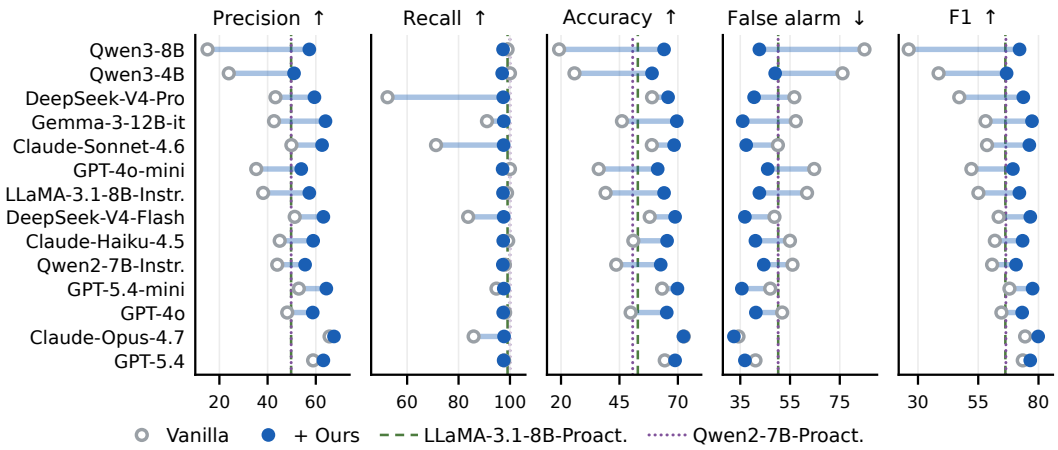


Figure 4: One graph controller improves F1 across all 14 downstream backbones. Open and filled markers show vanilla and + *Ours*; vertical reference lines show the two published fine-tuned proactive baselines. The mean F1 gain is 16.7 points; on the two corresponding backbones, + *Ours* also exceeds the published fine-tuned baselines. All backbones use the same TGL model and threshold. Vanilla values combine published legacy results and our recent-model runs; Table 5 gives their provenance and exact values.

5.1 Setup

We evaluate proactive triggering and routing on the desktop ProactiveAgent benchmark using its released splits and reward-model judge (Lu et al., 2025). The judge scores generated task suggestions under the benchmark's precision, recall, accuracy, false-alarm, and F1 definitions. We use F1 to assess the complete suggestion pipeline and precision and recall to distinguish false alarms from missed opportunities. Vanilla results use the originating paper's prompt protocol; + *Ours* supplies the frozen downstream language agent with the graph controller's trigger and routed context. Appendix A evaluates the graph-encoder design on mobile intent prediction with FingerTip-20K (Yang et al., 2026).

We use the same TGL model, trigger threshold, and prompt template across all backbones. We evaluate 14 instruction-following backbones spanning four families (open-weight, OpenAI, Anthropic, DeepSeek) as the downstream language agent, with two fine-tuned proactive baselines from Lu et al. (2025) for reference. Backbone list, model architecture, prompt, serving setup, and graph-construction rules are deferred to Appendices B, C, and F.

5.2 Main Results

The controller selects intervention times and entities independently of the downstream backbone; the language agent determines the resulting suggestion. We therefore evaluate the same controller with 14 backbones to test its usefulness across different generators. Figure 4 compares each backbone with TGL against its vanilla configuration under the protocol of Lu et al. (2025). Appendix D.4 gives every metric and identifies the published and re-run vanilla values.

A shared controller improves every backbone. TGL raises F1 on all 14 downstream backbones by 3.1–46.0 points, with a mean gain of 16.7. Gains span open-weight models, including Qwen3-8B (+46.0) and LLaMA-3.1-8B (+17.0), and API models, including DeepSeek-V4-Pro (+26.6), Claude-Sonnet-4.6 (+17.6), and GPT-5.4 (+3.1). Every + *Ours* configuration uses the same trigger model, threshold, routing interface, and prompt. The controller can therefore be trained once and reused to improve different language agents, retaining the same control decisions as the downstream backbone changes.

Comparison with proactive fine-tuning. Proactive behavior can be learned by adapting the language agent or training a separate controller. We compare these choices on corresponding backbones to examine where to place task-specific training. The proactive models published by Lu et al. (2025),

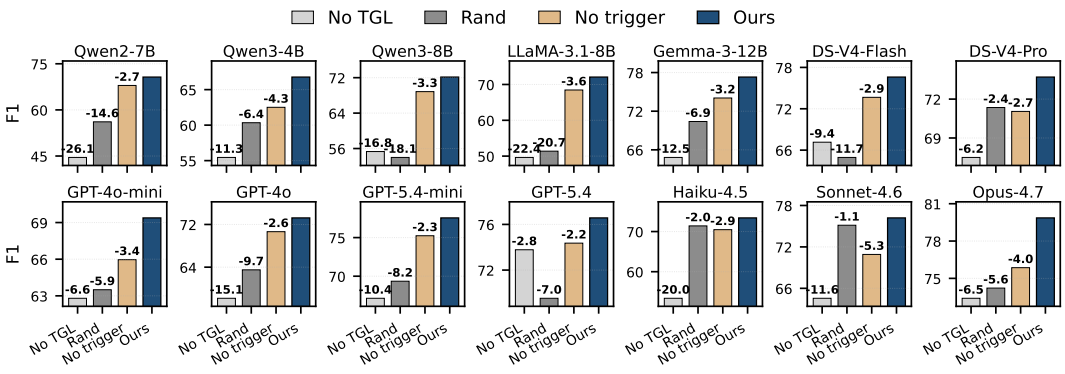


Figure 5: TGL improves F1 over all three controls on every backbone. Each panel compares *Ours* with prompt-only self-gating (No TGL), random routing (Rand), and generation at every event (No trigger).

LLaMA-3.1-8B-Proactive and Qwen2-7B-Proactive, reach F1 scores of 66.25 and 66.47. On the corresponding frozen instruction-tuned backbones, + *Ours* reaches 72.07 and 70.68, respectively. Training the 1.16 M-parameter controller thus achieves higher F1 than proactive fine-tuning of these two LLMs. Appendix D.3 further applies the trigger to the fine-tuned agents themselves: filtering their existing suggestions improves F1 for both, showing that the controller also complements prior proactive adaptation.

Higher precision with high recall. We examine precision and recall separately to characterize the F1 gains under the benchmark's evaluation protocol. Qwen3-8B starts with 99% recall and 15% precision; TGL raises its precision to 57% at 97% recall. DeepSeek-V4-Pro and Claude-Sonnet-4.6 start with recall of 52% and 71%; both exceed 97% with TGL. Across all backbones, precision improves and benchmark recall reaches at least 96%.

5.3 Ablation Study

We next examine the controller's two responsibilities: selecting intervention times and supplying entity context. The first two controls change invocation timing or routed content within the controller-guided pipeline; the third compares the complete interface with prompt-only self-gating. Figure 5 evaluates all three on the same 14 backbones. Appendix D.1 reports every per-backbone result.

Does the trigger matter? We first test the value of deciding when to invoke an agent that already receives routed context. *No trigger* invokes it on every event with the same routing list and prompt. Restoring the trigger improves mean F1 by 3.2 points, with gains of 2.2–5.3 across all 14 backbones. Selecting intervention times therefore improves suggestion quality, and the skipped events also save downstream calls (Section 5.5).

Does the routing content matter? The agent receives both session history and routed entities. To examine how the supplied entities affect its suggestions alongside that history, *Rand. routes* replaces each routing list with an equal-length list sampled from the training entity vocabulary. The history, trigger, and downstream prompt remain fixed. Using the learned routing list improves mean F1 by 8.6 points, with gains of 1.1–20.7 across all 14 backbones. The largest differences are on LLaMA-3.1-8B (20.7), Qwen3-8B (18.1), and Qwen2-7B (14.6). Graph-selected anchors consistently outperform unrelated entities supplied through the same interface.

Comparison with prompt-only self-gating. Output-style instructions also guide task formulation. We therefore compare the complete controller interface with an LLM that decides when and what to suggest under the same style requirements. The prompt-only baseline *No TGL* retains the output-style directives: a single sentence, at most 150 characters, led by a concrete deliverable. It asks the agent to choose between a task and an empty proposed-task string, omitting the routing field, routing-anchor instructions, and three routing demonstrations. *Ours* improves mean F1 by 12.7 points, with gains of 2.8–26.1 across all backbones. The controller-guided interface produces better suggestions than language-agent self-gating under the same output-style requirements.

Table 1: TGL achieves the highest trigger AUCs with lower latency than the LLM triggers. Architectures are trained from the ProactiveAgent release. AUC uses the three binary labelings in Section 5.4; Brier and ECE use the calibration target described in Appendix E.2. Trig. std describes the spread of F1-maximising thresholds across backbones. LLM latency includes single-forward and generation-derived protocols.

		Trigger AUC ↑			**Calibration** ↓		**Thresholds**	**Latency** (ms) ↓	
Trigger	Type	m-fire	c-skip	m-skip	Brier	ECE	Trig. std	Server	Local
Rule	Rule	0.585	0.581	0.496	0.319	0.294	0.143	0.0003	0.0001
LR (65 features)	Tabular	0.580	0.534	0.499	0.270	0.201	0.115	0.13	0.03
HGB (65 features)	Tabular	0.636	0.551	0.500	0.309	0.267	0.086	7.29	5.73
BGE-frozen + MLP	Textual	0.574	0.518	0.460	0.601	0.611	0.358	6.59	11.95
BGE-FT + MLP	Textual	0.621	0.586	0.553	0.593	0.599	0.396	6.86	11.06
Qwen3-0.6B	LLM	0.668	0.582	0.504	0.584	0.596	0.259	40.4	162.3
Qwen3-8B	LLM	0.644	0.618	0.597	0.580	0.592	0.222	78.6	1156.8
Proactive-Qwen3-0.6B (SFT)	LLM	0.505	0.508	0.485	0.657	0.657	0.140	3,927	11,966
TGL (Ours)	Graph	0.738	0.658	0.639	0.308	0.334	0.035	11.13	13.99

5.4 Trigger Architecture Comparison

Choosing an always-on trigger requires balancing decision quality with the cost of repeated evaluation. Table 1 compares nine architectures across five families—Rule, Tabular, Textual, LLM, and Graph—on trigger ranking and inference cost. Rule and tabular models test prediction from inexpensive event cues. Text classifiers test semantic encoding with the same BGE encoder used by TGL, including frozen and fine-tuned variants to examine the effect of encoder adaptation. The LLM rows cover both direct binary prediction and deriving a decision from task generation.

LR and HGB use event features; BGE-frozen + MLP and BGE-FT + MLP encode the latest five event texts; the TGL encoder processes the event–entity session graph. The LLM family includes fine-tuned Qwen3-0.6B and Qwen3-8B with a yes/no logit readout, plus Proactive-Qwen3-0.6B with a trigger derived from its generated task. Training data come from the same ProactiveAgent release, with event-label supervision for trigger classifiers and an additional entity-label objective for TGL. For downstream evaluation, each trigger supplies its decision to the same TGL routing list, prompt, and language agents. Appendix D.2 specifies the inputs and training recipe for each architecture.

Trigger evaluation labels. A trigger must recognise opportunities for assistance and occasions to remain silent. We assess their ranking with three binary labelings of the released test annotations. The *must-fire* set contains events requiring a non-null task ($\sim$24%). The *can-skip* set contains events accepting an empty suggestion ($\sim$51%); its *must-skip* subset contains events rejecting every alternative suggestion ($\sim$14% of test events). The three AUC columns use, respectively, membership in must-fire, membership outside can-skip, and membership outside must-skip as the positive label. Each column evaluates a separate target, with higher scores indicating a preference to fire.

TGL leads every trigger AUC. TGL reaches 0.738 AUC on must-fire ranking, +7.0 pp above the strongest LLM trigger (Qwen3-0.6B at 0.668). The two skip-side columns show the same pattern: TGL at 0.658 on can-skip and 0.639 on must-skip, +4.0 and +4.2 pp above the next best (Qwen3-8B at 0.618 and 0.597). The AUC lead comes with lower latency: TGL runs at 11.13 ms per event versus 40.4–3,927 ms for the three LLM-as-trigger configurations on the same server. The graph controller combines better ranking of fire and skip decisions with lower monitoring cost.

Deciding before generation reduces monitoring cost. The LLM rows also expose the cost of deriving a trigger from a generated task. The two LoRA Qwen3 triggers produce a yes/no score in one forward pass, costing 40.4 and 78.6 ms per event on the server. Proactive-Qwen3-0.6B generates a task string and maps a non-empty task to fire. Its mean 139 generated tokens cost 3,927 ms on the server and 11,966 ms on the laptop; the corresponding trigger AUCs are 0.505, 0.508, and 0.485. Predicting the intervention decision before composing a task removes decoding from the continuous

monitoring path. TGL implements this separation with a single forward pass that also supplies the routing list.

A shared threshold supports backbone changes. Reusing the controller across language agents also raises the question of whether its operating threshold needs adjustment. We use $\tau = 0.5$ for all 14 backbones and evaluate both sensitivity to this setting and transfer between backbones. On five open-weight backbones with three runs each, mean F1 ranges from 70.4 to 73.1 over $\tau \in [0.4, 0.6]$, with 72.1 at the deployed threshold (Appendix E.2). We also choose a threshold on four backbones and transfer it to the fifth. This incurs a mean loss of 0.4 F1 relative to that backbone's tuned threshold. The controller can therefore reuse its operating threshold across these backbones with little loss in F1. Table 1 additionally reports Brier score, ECE, and the across-backbone spread of F1-maximising thresholds; Appendix E.2 reports temperature calibration, and Figure 6 shows the score distributions.

5.5 Efficiency

The controller runs at every observation, including those that produce no suggestion, and keeps its weights resident between checks. We therefore measure both per-event latency and the full memory footprint, including the text encoder. An NVIDIA RTX A6000 (48 GB) and an Apple M3 Pro (36 GB) cover server and consumer-laptop deployment (Appendix F.9).

Per-event latency. TGL runs at 11.13 ms on the server. BGE encoding takes ∼7–8 ms for the event text and a fresh entity encoded together, and the graph layers add ∼3 ms. Recurring entities reuse cached embeddings, allowing the graph to retain their context with little additional encoding work. The Qwen3-0.6B and Qwen3-8B single-forward triggers take 40.4 and 78.6 ms, giving TGL a ∼4–7× trigger-stage speedup.

Consumer-laptop deployment. The same TGL model runs at 13.99 ms per event on the M3 Pro. Qwen3-0.6B and Qwen3-8B take 162.3 and 1,156.8 ms, respectively, giving trigger-stage speedups of ∼12× and ∼83×. The Tabular, Textual, and Graph triggers all run within 14 ms on this laptop. TGL combines the highest trigger AUC with sub-14 ms inference on this consumer device.

Memory footprint. TGL has 1.16 M trainable parameters and a 109 M frozen BGE encoder. BF16 resident weights occupy ∼220 MiB, and streaming inference peaks at ∼267 MiB. The Qwen3-0.6B and Qwen3-8B triggers require approximately 1.5 GB and 16 GB of BF16 weights, or ∼7–75× the controller's resident footprint (Table 13). The controller, including its text encoder, occupies a small share of laptop memory and leaves room for the user's applications.

Downstream generation. Graph prediction determines the cost of each monitoring check; triggered events additionally incur a language-agent call. Appendix D.5 reports the wall-clock cost of each backbone's generation call. The *No trigger* ablation in Section 5.3 evaluates the effect of invocation timing on suggestion quality.

6 Conclusion

Continuous proactive assistance requires inexpensive decisions about when to intervene and which context should guide the intervention. We have shown that user activity's native event–entity structure supports both decisions through a small graph model. TGL links events through shared entities and jointly learns event-level triggering and entity-level context selection, producing both outputs in one forward pass. It achieves the highest AUCs among nine trigger architectures with ∼ 4–7× trigger-stage speedups over the two single-forward LLM triggers. A shared TGL model improves F1 across all 14 downstream backbones by a mean of 16.7 points and runs at 13.99 ms per event on a consumer laptop. These results establish a practical design for proactive agents: a compact graph model handles continuous monitoring and context selection, and a language agent turns the selected context into user-facing assistance.

Ethics Statement

This work uses the publicly released ProactiveAgent and FingerTip-20K benchmarks under their original terms (Lu et al., 2025; Yang et al., 2026). The experiments use the released activity records,

and our code release contains the graph-construction and modelling pipeline. The study develops an efficient trigger and context-selection controller for proactive agents.

Always-on activity logs are privacy-sensitive: file names, URLs, search queries, application switches, and (on FingerTip) demographic profile fields all flow through the trigger and routing heads. The graph controller's small footprint supports on-device activity processing (Section 5.5). Deployment must also account for the activity context sent to the downstream language agent. Production deployment additionally requires standard data minimisation, sensitive-entity filtering, opt-out, and retention controls. Shared-device deployments require separate per-user graphs and access controls. A badly-timed suggestion interrupts focused work and repeated mistimed interventions erode trust. The trigger head is trained to suppress such events, and a deployment should monitor this failure mode directly. Fairness analysis across the FingerTip demographic subgroups (gender, age, occupation, location, device) is required before any are used as routing features in a deployed system.

Reproducibility Statement

We evaluate on ProactiveAgent with its released reward-model judge (Lu et al., 2025) and on FingerTip-20K with its released splits and metrics (Yang et al., 2026). Table 5 identifies published reference values and our evaluation runs.

Appendix B specifies graph extraction, node types, edge relations, and label derivation. Appendix F gives the encoder, heads, training settings, inference procedure, sampling settings, and latency protocol. The downstream system prompts appear in Appendix G. Appendix E reports retraining variation, threshold sensitivity, and construction variants. Code and the graph-construction pipeline will be released.

AI Use Statement

We use LLMs as general research assistants for literature discovery, feedback on experimental design, code implementation and debugging, result analysis, and manuscript drafting and editing. The authors review AI-assisted outputs and take responsibility for the paper, code, and reported results.

References

Piotr D Adamczyk and Brian P Bailey. If not now, when? the effects of interruption at different moments within task execution. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 271–278. ACM, 2004. URL https://experts.illinois.edu/en/publications/if-not-now-when-the-effects-of-interruption-at-different-moments-/.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=F72ximsx7C1.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

Dongping Chen, Yue Huang, Siyuan Wu, Jingyu Tang, Huichi Zhou, Qihui Zhang, Zhigang He, Yilin Bai, Chujie Gao, Liuyi Chen, Yiqiang Li, Chenlong Wang, Yue Yu, Tianshuo Zhou, Zhen Li, Yi Gui, Yao Wan, Pan Zhou, Jianfeng Gao, and Lichao Sun. GUI-World: A video benchmark and dataset for multimodal GUI-oriented understanding. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/3926df1a00c9abf056df7bcf253d026a-Abstract-Conference.html.

Valerie Chen, Alan Zhu, Sebastian Zhao, Hussein Mozannar, David Sontag, and Ameet Talwalkar. Need help? designing proactive AI assistants for programming. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–18, 2025b. doi: 10.1145/3706598.3714002. URL https://doi.org/10.1145/3706598.3714002.

Mary Czerwinski, Eric Horvitz, and Susan Wilhite. A diary study of task switching and interruptions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 175–182. ACM, 2004. doi: 10.1145/985692.985715.

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pp. 845–854, 2017. doi: 10.1145/3126594.3126651. URL https://doi.org/10.1145/3126594.3126651.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, pp. 28091–28114, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/5950bf290a1570ea401bf98882128160-Abstract-Datasets_and_Benchmarks.html.

Yang Deng, Lizi Liao, Zhonghua Zheng, Grace Hui Yang, and Tat-Seng Chua. Towards human-centered proactive conversational agents. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 807–818, 2024. doi: 10.1145/3626772.3657843. URL https://doi.org/10.1145/3626772.3657843.

Christopher Diebel, Marc Goutier, Martin Adam, and Alexander Benlian. When AI-based agents are proactive: Implications for competence and system satisfaction in human–AI collaboration. *Business & Information Systems Engineering*, 68(2):259–278, 2026. doi: 10.1007/s12599-024-00918-y. URL https://doi.org/10.1007/s12599-024-00918-y.

Lei Ding, Bin He, Chenguang Wang, and Yang Liu. ProActor: Timing-aware reinforcement learning for proactive task scheduling agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18257–18303. Association for Computational Linguistics, 2026a. doi: 10.18653/v1/2026.acl-long.832. URL https://aclanthology.org/2026.acl-long.832/.

Zhijie Ding, Weinan Hong, Zicheng Zhu, Lei Li, Dezhi Kong, Hao Wang, Peng Zhou, Xuchu Jiang, and Jiaming Xu. Perceive before reasoning: A pre-reasoning perception framework for efficient and reliable proactive mobile agents. *arXiv preprint arXiv:2606.03236*, 2026b. URL https://arxiv.org/abs/2606.03236.

Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, Hengxu Wang, Luowei Zhou, and Mike Zheng Shou. AssistGUI: Task-oriented PC graphical user interface automation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13289–13298, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Gao_AssistGUI_Task-Oriented_PC_Graphical_User_Interface_Automation_CVPR_2024_paper.html.

Jennifer Gluck, Andrea Bunt, and Joanna McGrenere. Matching attentional draw with utility in interruption. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 41–50, 2007. doi: 10.1145/1240624.1240631.

Sitong Gong, Tianyu Yan, Caixin Kang, Bo Zheng, Xiang Ruan, Huchuan Lu, Kaipeng Zhang, Yoichi Sato, and Yifei Huang. Vinci2: Providing proactive assistance in continuous egocentric videos. In *European Conference on Computer Vision*, 2026. URL https://arxiv.org/abs/2607.11523.

Jongyi Hong, Eui-Ho Suh, Junyoung Kim, and Su-Yeon Kim. Context-aware system for proactive personalized service based on context history. *Expert Systems with Applications*, 36(4):7448–7457, 2009. doi: 10.1016/j.eswa.2008.09.002. URL https://doi.org/10.1016/j.eswa.2008.09.002.

Yizheng Huang and Jimmy Xiangji Huang. A survey on retrieval-augmented text generation for large language models. *ACM Computing Surveys*, 58(12):1–38, 2026. doi: 10.1145/3805774. URL https://doi.org/10.1145/3805774.

Shamsi T. Iqbal and Brian P. Bailey. Effects of intelligent notification management on users and their tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 93–102, 2008. doi: 10.1145/1357054.1357070. URL https://doi.org/10.1145/1357054.1357070.

Shamsi T Iqbal and Eric Horvitz. Disruption and recovery of computing tasks: field study, analysis, and directions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 677–686. ACM, 2007. URL https://erichorvitz.com/chi_2007_iqbal_horvitz.pdf.

Woojeong Jin, Meng Qu, Xisen Jin, and Xiang Ren. Recurrent event network: Autoregressive structure inference over temporal knowledge graphs. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pp. 6669–6683, Stroudsburg, PA, USA, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.541. URL https://doi.org/10.18653/v1/2020.emnlp-main.541.

Brennan Jones, Yan Xu, Qisheng Li, and Stefan Scherer. Designing a proactive context-aware AI chatbot for people's long-term goals. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pp. 1–7, 2024. doi: 10.1145/3613905.3650912.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. OmniACT: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *European Conference on Computer Vision*, pp. 161–178, 2024. doi: 10.1007/978-3-031-73113-6_10. URL https://doi.org/10.1007/978-3-031-73113-6_10.

Jiho Kim, Junseong Choi, Woosog Chay, Daeun Kyung, Yeonsu Kwon, Yohan Jo, and Edward Choi. ProPerSim: Developing proactive and personalized AI assistants through user-assistant simulation. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/b3bdc41d0b2387eabbb719956a38f295-Abstract-Conference.html.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, Toulon, France, 2017. OpenReview.net. URL https://openreview.net/forum?id=SJU4ayYgl.

Dezhi Kong, Zhengzhao Feng, Qiliang Liang, Hao Wang, Haofei Sun, Changpeng Yang, Yang Li, Peng Zhou, Shuai Nie, Hongzhen Wang, Linfeng Zhou, Hao Jia, Jiaming Xu, Runyu Shi, and Ying Huang. ProactiveMobile: A comprehensive benchmark for boosting proactive intelligence on mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 27503–27513, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Kong_ProactiveMobile_A_Comprehensive_Benchmark_for_Boosting_Proactive_Intelligence_On_Mobile_CVPR_2026_paper.html.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.nips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Guanzhen Li, Liangming Pan, and Leye Wang. ProEvent: An event-centric benchmark for proactive agents. *arXiv preprint arXiv:2607.17701*, 2026. URL https://arxiv.org/abs/2607.17701.

Hao Li, Chenghao Yang, An Zhang, Yang Deng, Xiang Wang, and Tat-Seng Chua. Hello again! LLM-powered personalized agent for long-term dialogue. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5259–5276, 2025. doi: 10.18653/v1/2025.naacl-long.272. URL https://aclanthology.org/2025.naacl-long.272/.

Yujia Li, Shiliang Sun, and Jing Zhao. TiRGN: Time-guided recurrent graph network with local-global historical patterns for temporal knowledge graph reasoning. In Luc De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pp. 2152–2158, San Francisco, CA, USA, 2022a. ijcai.org. doi: 10.24963/ijcai.2022/299. URL https://doi.org/10.24963/ijcai.2022/299.

Zixuan Li, Xiaolong Jin, Wei Li, Saiping Guan, Jiafeng Guo, Huawei Shen, Yuanzhuo Wang, and Xueqi Cheng. Temporal knowledge graph reasoning based on evolutional representation learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, pp. 408–417, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. doi: 10.1145/3404835.3462963. URL https://doi.org/10.1145/3404835.3462963.

Zixuan Li, Saiping Guan, Xiaolong Jin, Weihua Peng, Yajuan Lyu, Yong Zhu, Long Bai, Wei Li, Jiafeng Guo, and Xueqi Cheng. Complex evolutional pattern learning for temporal knowledge graph reasoning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pp. 290–296, Stroudsburg, PA, USA, 2022b. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.32. URL https://doi.org/10.18653/v1/2022.acl-short.32.

Tianjian Liu, Fanqi Wan, Jiajian Guo, and Xiaojun Quan. ProactiveEval: A unified evaluation framework for proactive dialogue agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 41068–41100, 2026. doi: 10.18653/v1/2026.acl-long.1906. URL https://aclanthology.org/2026.acl-long.1906/.

Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, Weiwen Liu, Yasheng Wang, Zhiyuan Liu, Fangming Liu, and Maosong Sun. Proactive agent: Shifting LLM agents from reactive responses to active assistance. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sRIU6k2TcU.

Matheus Kunzler Maldaner, Adam Fourney, Amanda Swearngin, Hussein Mozannar, Gagan Bansal, Maya Murad, Rafah Hosn, and Saleema Amershi. SentinelBench: A benchmark for long-running monitoring agents. *arXiv preprint arXiv:2606.05342*, 2026. URL https://arxiv.org/abs/2606.05342.

Gloria Mark, Daniela Gudith, and Ulrich Klocke. The cost of interrupted work: more speed and stress. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 107–110. ACM, 2008. URL https://www.ics.uci.edu/~gmark/chi08-mark.pdf.

Christian Meurisch, Cristina A. Mihale-Wilson, Adrian Hawlitschek, Florian Giger, Florian Müller, Oliver Hinz, and Max Mühlhäuser. Exploring user expectations of proactive AI systems. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(4):1–22, 2020. doi: 10.1145/3432193. URL https://doi.org/10.1145/3432193.

Karen Myers and Neil Yorke-Smith. Proactive behavior of a personal assistive agent. In *Proceedings of the AAMAS Workshop on Metareasoning in Agent-Based Systems, Honolulu, HI*, pp. 31–45, 2007. URL https://www.sri.com/publication/artificial-intelligence-pubs/proactive-behavior-of-a-personal-assistive-agent/.

Jean Oh, Felipe Meneguzzi, and Katia Sycara. Probabilistic plan recognition for intelligent information agents-towards proactive software assistant agents. In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pp. 281–287, 2011. URL https://www.scitepress.org/papers/2011/31871/31871.pdf.

Chris Parnin and Spencer Rugaber. Programmer information needs after memory failure. In *Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC)*, pp. 123–132, 2012. URL https://chrisparnin.me/pdf/infoneeds.pdf.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=il5yUQsrjC.

Chris Richardson, Yao Zhang, Kellen Gillespie, Sudipta Kar, Arshdeep Singh, Zeynab Raeesy, Omar Zia Khan, and Abhinav Sethy. Integrating summarization and retrieval for enhanced personalization via large language models, 2023. URL https://arxiv.org/abs/2310.20081.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020. URL https://arxiv.org/abs/2006.10637.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. LaMP: When large language models meet personalization. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7370–7392, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.399. URL https://aclanthology.org/2024.acl-long.399/.

Nitin Sawhney and Chris Schmandt. Nomadic radio: Scaleable and contextual notification for wearable audio messaging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 96–103, 1999. URL https://www.media.mit.edu/speech/old/publications.html.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam (eds.), *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pp. 593–607, Cham, CHE, 2018. Springer. doi: 10.1007/978-3-319-93417-4_38. URL https://doi.org/10.1007/978-3-319-93417-4_38.

Yunxiao Shi, Wujiang Xu, Zeqi Zhang, Xing Zi, Qiang Wu, and Min Xu. PersonaX: A recommendation agent-oriented user modeling framework for long behavior sequence. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5764–5787. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.300. URL https://aclanthology.org/2025.findings-acl.300/.

Raymond So and Liz Sonenberg. Situation awareness in intelligent agents: Foundations for a theory of proactive agent behavior. In *Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004.(IAT 2004).*, pp. 86–92. IEEE, 2004.

Weiwei Sun, Xuhui Zhou, Weihua Du, Xingyao Wang, Sean Welleck, Graham Neubig, Maarten Sap, and Yiming Yang. Training proactive and personalized LLM agents. In *Conference on Language Modeling*, 2026. URL https://arxiv.org/abs/2511.02208.

Zhaoxuan Tan, Qingkai Zeng, Yijun Tian, Zheyuan Liu, Bing Yin, and Meng Jiang. Democratizing large language models via personalized parameter-efficient fine-tuning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 6476–6491, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.372. URL https://aclanthology.org/2024.emnlp-main.372/.

Yuanbo Tang, Huaze Tang, Tingyu Cao, Lam Nguyen, Anping Zhang, Xinwen Cao, Chunkang Liu, Wenbo Ding, and Yang Li. ProAgentBench: Evaluating LLM agents for proactive assistance with real-world data. *arXiv preprint arXiv:2602.04482*, 2026. URL https://arxiv.org/abs/2602.04482.

Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-Evolve: Deep temporal reasoning for dynamic knowledge graphs. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th*

International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of *Proceedings of Machine Learning Research*, pp. 3462–3471, Cambridge, MA, USA, 2017. PMLR. URL http://proceedings.mlr.press/v70/trivedi17a.html.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, Addis Ababa, Ethiopia, 2020. OpenReview.net. URL https://openreview.net/forum?id=BylA_C4tPr.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, Vancouver, BC, Canada, 2018. OpenReview.net. URL https://openreview.net/forum?id=rJXMpikCZ.

Yufeng Wang, Wei Zhang, Zhiquan Wen, Jinwu Hu, Linhui Xiao, Tianlu Pan, Qingfang Zheng, and Mingkui Tan. Preference-driven online adaptation for personalized interaction initiation in proactive AI assistants. *arXiv preprint arXiv:2608.04416*, 2026. URL https://arxiv.org/abs/2608.04416.

Zixuan Wang, Bo Yu, Junzhe Zhao, Wenhao Sun, Sai Hou, Shuai Liang, Xing Hu, Yinhe Han, and Yiming Gan. KARMA: Augmenting embodied AI agents with long-and-short term memory systems. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2025. doi: 10.1109/ICRA55743.2025.11128047. URL https://doi.org/10.1109/ICRA55743.2025.11128047.

Wolfgang Woerndl, Johannes Huebner, Roland Bader, and Daniel Gallego-Vico. A model for proactivity in mobile, context-aware recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, pp. 273–276, 2011. doi: 10.1145/2043932.2043981. URL https://doi.org/10.1145/2043932.2043981.

Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey. *ACM Computing Surveys*, 55(5):1–37, 2023. doi: 10.1145/3535101. URL https://doi.org/10.1145/3535101.

Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 346–353, 2019. URL https://arxiv.org/abs/1811.00855.

Zhifei Xie, Zongzheng Hu, Fangda Ye, Xin Zhang, Haobo Chai, Zihang Liu, Pengcheng Wu, Guibin Zhang, Yue Liao, Xiaobin Hu, Deheng Ye, Chunyan Miao, and Shuicheng Yan. PASK: Toward intent-aware proactive agents with long-term memory. *arXiv preprint arXiv:2604.08000*, 2026. URL https://arxiv.org/abs/2604.08000.

Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S. Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. Graph contextualized self-attention network for session-based recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pp. 3940–3946, 2019.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/2002.07962.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning (ICML)*, pp. 5453–5462, 2018. URL https://proceedings.mlr.press/v80/xu18c.html.

Bufang Yang, Lilin Xu, Liekang Zeng, Yunqi Guo, Siyang Jiang, Wenrui Lu, Kaiwei Liu, Yixuan Li, Xiaofan Jiang, Guoliang Xing, and Zhenyu Yan. ProAgent: Harnessing on-demand sensory contexts for proactive LLM agent systems in the wild. *arXiv preprint arXiv:2512.06721*, 2025a. URL https://arxiv.org/abs/2512.06721.

Bufang Yang, Lilin Xu, Liekang Zeng, Kaiwei Liu, Siyang Jiang, Wenrui Lu, Hongkai Chen, Xiaofan Jiang, Guoliang Xing, and Zhenyu Yan. ContextAgent: Context-aware proactive LLM agents with open-world sensory perceptions. In *Advances in Neural Information Processing Systems*, volume 38, pp. 167509–167543, 2025b. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f4e5cd2079f5a6cf5176818add841531-Abstract-Conference.html.

Qinglong Yang, Haoming Li, Haotian Zhao, Xiaokai Yan, Jingtao Ding, Fengli Xu, and Yong Li. FingerTip 20k: A benchmark for proactive and personalized mobile LLM agents. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2507.21071.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, Feng Yin, Yitao Liang, and Yaodong Yang. Proagent: building proactive cooperative agents with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17591–17599, 2024a.

Xuan Zhang, Yang Deng, Zifeng Ren, See-Kiong Ng, and Tat-Seng Chua. Ask-before-plan: Proactive language agents for real-world planning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10836–10863, 2024b. doi: 10.18653/v1/2024.findings-emnlp.636. URL https://aclanthology.org/2024.findings-emnlp.636/.

Yichi Zhang, Xin Luna Dong, Zhaojiang Lin, Andrea Madotto, Anuj Kumar, Babak Damavandi, Joyce Chai, and Seungwhan Moon. Proactive assistant dialogue generation from streaming egocentric videos. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 12044–12068, 2025. doi: 10.18653/v1/2025.emnlp-main.605. URL https://aclanthology.org/2025.emnlp-main.605/.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

Yuyang Zhao, Wentao Shi, Fuli Feng, and Xiangnan He. AppAgent-Pro: A proactive GUI agent system for multidomain information integration and user assistance. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 6767–6771, 2025. doi: 10.1145/3746252.3761473. URL https://doi.org/10.1145/3746252.3761473.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

A Graph-Based Mobile Intent Prediction

To examine graph-based context selection for mobile intent prediction, we apply the temporal graph-encoder design to FingerTip-20K (Yang et al., 2026). Each episode supplies a graph of user context, historical activity, and categorical entities. We train the TGL encoder from scratch for 40 epochs on FingerTip's training split and use its embedding of the current episode to retrieve training intents. We retain candidates for the application named by the highest-ranked retrieved intent before passing them to the downstream `Qwen2.5-VL-7B-Instruct` agent (Appendices B.2 and C.2).

A.1 Evaluation Protocol

We adopt the same evaluation as FingerTip-20K. The proactive task-suggestion track formalises the task as $I = f(U, T, S, I_{\text{history}}, O)$, where the agent predicts a Chinese natural-language intent I from the user profile U, the timestamp T, the scenario (location category) S, the user's last 20 historical intents I_{history} drawn from the global usage log, and t initial screenshots O from the current episode ($t \in \{0, 1, 2, 3\}$ denotes the number of screenshots). We evaluate on the public `Test-suggestion` split and report two metrics: intent similarity, $\text{Sim}_1 = (S_1 + S_2)/2$, where S_1 is the `paraphrase-multilingual-MiniLM-L12-v2` cosine similarity and S_2 is the Levenshtein similarity, both normalised to $[0, 1]$; and intent success rate SR_1, the percentage of predicted intents judged equivalent to their targets by DeepSeek-V3.

A.2 Results

Table 2: Mobile intent prediction on FingerTip-20K's public `Test-suggestion` split. Intent similarity (Sim_1) averages embedding and edit-distance similarity; success rate (SR_1) is judge-assessed intent equivalence. The first five rows are published baselines from Yang et al. (2026). Our rows use frozen Qwen2.5-VL-7B and the last 20 historical intents. LoRA denotes low-rank adaptation; rank is the adapter dimension.

Method and image input	Intent similarity $\text{Sim}_1 \uparrow$	Success rate SR_1 (%) $\uparrow$
Qwen2.5-VL-7B (0 screenshots)	0.25	3.1
GPT-4.1 (0 screenshots)	0.35	7.2
GPT-4.1 (3 screenshots, full history)	0.55	13.8
Qwen2.5-VL-7B fine-tuned (LoRA rank 4)	0.52	20.3
Qwen2.5-VL-7B fine-tuned (LoRA rank 64)	0.55	26.0
TGL + Qwen2.5-VL-7B (2 screenshots)	**0.61**	**26.5**
TGL + Qwen2.5-VL-7B (3 screenshots)	0.59	26.2

Table 2 compares against the published numbers from Yang et al. (2026). TGL plus Qwen2.5-VL-7B with two screenshots exceeds the paper's strongest model fine-tuned with low-rank adaptation (LoRA) on both metrics ($\text{SR}_1 = 26.5\%$ vs. 26.0%, $\text{Sim}_1 = 0.61$ vs. 0.55). With three screenshots, our system exceeds the published GPT-4.1 baseline using three screenshots and full history: we improve Sim_1 by $+0.04$ and SR_1 by $+12$ percentage points using only the last 20 history events. The graph-based retrieval pipeline provides contextual intent candidates to the frozen Qwen2.5-VL-7B agent.

B Per-Session Graph Construction

Activity structure and graph schema. User activity links events to persistent entities such as applications, files, web addresses, and search queries. Our graph schema represents these entities as typed nodes, connects them to the events that involve them, and records event order through temporal edges. Platform metadata supplies application names, file types, URL domains, and timestamps. The benchmark implementations map their released activity records into this schema.

Shared representation. Each graph contains activity nodes, entity nodes, and temporal edges linking activities in order. Desktop activity nodes represent individual events. A mobile graph contains historical activity nodes and one *current-episode node*, whose representation predicts the user's next intent. Entity nodes represent named objects such as applications and files. Additional *category nodes* represent properties shared by many objects, such as a file extension or an intent class. Every node has a frozen text embedding, a learned node-type embedding, and an embedding of its time offset. The three-layer graph neural network (GNN) combines these features through message passing (Appendix F). Each platform supplies its own node and relation vocabulary and prediction heads.

Entity extraction. The desktop implementation applies regex patterns and categorical mappings to ProactiveAgent's released event descriptions. ProactiveAgent constructs these descriptions from collected and simulated activity, including language-model conversion of structured records to text (Lu

et al., 2025). The extractor identifies applications, files, web addresses, search queries, and quoted artifacts, and the full event text supplies the event-node embedding. FingerTip provides categorical episode fields that map directly to typed nodes. These two input formats share the event–entity graph representation described below.

B.1 Instantiation: Desktop Activity Logger (ProactiveAgent)

We construct each desktop session graph with deterministic entity extraction, category-node construction, and temporal linking.

Inputs

For each session we receive a sequence of timestamped event strings from the released ProactiveAgent data. Examples: "*The user opened 'email_filter.py' in Visual Studio Code.*", "*The user searched for 'js fade-in' on Bing.*". Each session forms an independent graph. Metadata-only events (lines starting with "`# Assistant Available Operations`" or containing "`Operation(name=`") are filtered out before extraction.

Entity Extraction

The extractor identifies five entity classes and adds shared category nodes:

(i) **Applications.** Alias lookup maps names to eight canonical application labels: Visual Studio Code, IntelliJ IDEA, Outlook, Jupyter Notebook, Terminal, Browser, Email Client, and IDE. For example, Visual Studio Code, VSCode, and Code.exe map to the same application node; Chrome, Bing, and Google map to Browser.

(ii) **Files.** A recognised filename extension identifies a file. Each file also contributes two category nodes: its *file extension* (e.g., Python's `.py`) and its *file category* (code, writing, data, slides, or other). The extension inventory contains 25 extensions observed in the training stream and grouped using standard file-type conventions.

(iii) **Web addresses (URLs).** A string beginning with `http://` or `https://` identifies a web address. Its *website-domain category* is obtained by stripping a leading `www.` and retaining the final two dot-separated hostname components.

(iv) **Search queries.** Search and typing patterns, such as "searched for …" or "typed …", identify the query text. A shared *query-language category* groups the text as Chinese, English, mixed, or other using its Unicode script. This heuristic assigns romanised Chinese written in Latin letters to the English category.

(v) **Document artifacts.** A quoted title or name is extracted when preceded by a document noun: file, document, report, spreadsheet, article, tab, summary, email, or draft. Matches are merged with filename matches, so "document 'report.docx"' yields one file node.

Application aliases and extraction patterns are fixed from the training stream. Entities with the same type and lowercased surface label share a node within a session. Stable identifiers combine the entity type with the first 16 hexadecimal characters of the label's SHA-1 hash. Graph instances are created separately for each session. The code supplies the full application-alias and extension-to-category mappings.

Table 3 summarises the rules. For example, "The user opened 'email_filter.py' in Visual Studio Code" contributes two named entities—the file and application—and two category nodes—Python extension and code file. Both named entities and category nodes are connected to the event and are eligible for routing. Appendix E.3 tests the effect of removing these node groups.

Event Records and Encoder Edges

The extraction record stores each event, its entities, and their order. It marks the first application mentioned in an event as its *primary entity*, falling back to the first extracted entity when no application is present. Links between consecutive primary entities preserve interaction provenance in the extraction record. The encoder reads five relation channels: event-to-entity, entity-to-event, forward

Table 3: Desktop graph construction. Named entities are extracted from event text; shared category nodes encode their file, website, or language properties.

Entity class	Extraction rule	Added category nodes
Application	Application-name aliases	–
File	Recognised filename extension	File extension; file category
Web address	Web-address syntax	Website domain
Search query	Search or typing pattern	Query-language category
Document artifact	Quoted name after a document noun	–

temporal, backward temporal, and self-loop. Forward temporal edges connect an earlier event to the next event; backward edges connect the later event to the earlier one. Each relation has a learned embedding.

Training uses both temporal directions. At inference, backward temporal edges are removed and future-node input representations are masked according to Appendix F.7. Appendix E.1 evaluates this training choice. Each event node records the elapsed time since its predecessor; entity and category nodes have time offset $\Delta t = 0$. The routing head ranks the entity and category nodes connected to the current event. Its highest-scoring node is the *routing anchor* supplied to the language agent.

Trigger and Entity-Relevance Labels

Each session in the released ProactiveAgent training set (Lu et al., 2025) comes with a (`history`, `current_observation`, `proposed_task`) triple where `proposed_task` is either a non-null task suggestion or `null`. We derive supervision as follows.

Trigger label on the event node: 1 iff the assistant proposed a non-null task at this event, else 0. The benchmark also marks acceptance in-line via the substring "`Is Accepted: True`", which forces label 1 on the corresponding event even when the assistant turn is null at that turn.

Routing labels on entity nodes. We apply the same extractor to the proposed-task text and match its surface forms to entities in the event graph. Matches supply positive routing labels and the other entities at that event supply negative labels. A positive-trigger event with an empty match set contributes trigger supervision alone. Negative-trigger events supply negative labels for their associated entities. The session-graph loader assigns one training label to each entity node; if that entity has multiple label-table entries within a session, the last entry supplies its training target. At inference, entity scores are recomputed for the current event history and the routing list is restricted to nodes connected to that event.

The comparison uses the same released training-supervision source across the learned trigger rows and the proactive-finetuning baselines, while evaluation inputs are the held-out event histories.

Train/Test Isolation

ProactiveAgent provides separate training and evaluation sessions. Category-level entities, including application names and file-type nodes, recur across sessions and splits. The graph representation also includes session-specific file names, search queries, and document artifacts. Each session is processed independently. An event-text matching check found that 0% of evaluation event strings also occur in the training set.

B.2 Instantiation: Mobile Usage Logger (FingerTip)

FingerTip provides application, intent-class, scenario, and time fields for each episode, along with a user profile containing gender, age, occupation, location, and device. Applications use Android package identifiers; a name lookup built from training intents supplies Chinese labels such as 微信 (WeChat). Intent classes follow the benchmark's 42-class taxonomy. Each episode yields one graph with seven node types and eight edge relations.

Node Types

(i) **Current episode:** one node containing the current time, scenario, and user profile. This node's representation is used for prediction. Its Chinese text concatenates these fields with the labels 时间 (time), 场景 (scenario), and 用户画像 (user profile).

(ii) **Historical activity:** one node for each of the user's last 20 intents in the released usage log. Its text contains that activity's time, scenario, application, intent class, and intent description, introduced by the Chinese labels 时间, 场景, App, 类, and 意图.

(iii) **Application:** one node per distinct application in the history, using the Chinese name obtained from the training-set lookup.

(iv) **Intent class:** one node per distinct class in the history, such as alarm/timer, video, or shopping.

(v) **Scenario:** one node per location category, such as home, workplace, or transit.

(vi) **Hour of day:** one node per distinct hour, retaining the hour in its text and adding one of five Chinese time-of-day labels: late night (0–5), morning (6–11), noon (12–13), afternoon (14–17), or evening (18–23).

(vii) **User profile:** one node containing the profile supplied for the user. Entity and category nodes are reused within the episode graph when their type and value match.

Edge Relations

Five entity relations connect activity nodes to applications, intent classes, scenarios, hours, and user profiles. The current-episode node connects to its profile, scenario, and hour. Each historical-activity node connects to its application, intent class, scenario, and hour. These entity links carry the same relation label in both directions. Two temporal relations connect consecutive historical activities: earlier-to-later and later-to-earlier. They also link the most recent activity to the current episode in both directions. A self-loop relation lets every node retain its own features during attention. The same history graph is used during training and inference.

Heads and Training

The current-episode node's final representation drives the prediction. Three heads sit on it: a 462-class app classifier, a 42-class intent-class classifier, and a 1024-d semantic-alignment head whose target is the BGE embedding of the target intent text; the semantic head supplies the retrieval vector at inference, and the two classifiers provide auxiliary training losses. The text encoder is `BAAI/bge-large-zh-v1.5` (1024d) for Chinese text. The model is trained for 40 epochs on FingerTip's released training split with AdamW (learning rate 3×10^{-4}, weight decay 10^{-4}), batch 32 graphs. Total trainable parameters are ≈ 2.5 M. These heads implement FingerTip's episode-level intent prediction task.

C Downstream Language Agent: Design Details

Shared design. A frozen instruction-tuned language model receives structured context and few-shot demonstrations, then emits a JSON object containing the proactive suggestion. The desktop implementation serves all 14 downstream backbones in Table 5. Local vLLM inference uses temperature 0, and API-served backbones use their default sampling settings.

Per-platform inputs. The desktop agent receives the rolling session history, current observation, and scored routing entities. The FingerTip agent receives the user profile, scenario, time, latest 20 historical intents, retrieved candidates, and episode screenshots.

Output formats. We generate an English task suggestion for ProactiveAgent and a Chinese intent string for FingerTip. The following subsections specify our prompts, length limits, decoding settings, and parsing procedures.

C.1 Instantiation: Desktop Agent (ProactiveAgent)

The desktop language agent generates a task suggestion from the current observation, recent history, and controller output.

Inputs and Output Schema

For each event the agent receives three inputs: *(i)* the rolling session history, the last five raw event texts in the current session (appended at every event regardless of whether the agent fires, so non-fire events still contribute context to subsequent fire events); *(ii)* the current observation, the event text at step t describing what the user is doing right now; and *(iii)* the routed nodes, the highest-scoring entities from the TGL routing head with their scores, e.g. `[{"node": "email_filter.py", "score": 0.78}, ...]`, sorted by descending s_{rout}.

The routing list provides named entities and their relevance scores. The output JSON contains a proposed-task string and five auxiliary fields: session theme, purpose, thoughts, response, and intent. The proposed-task field is evaluated by the benchmark; the auxiliary fields are logged. An empty task string represents abstention. We use structured-output decoding on backends that support the schema.

System Prompt

In every prompt and prompt example shown in this appendix and in Appendix G, the token "TGN" is used as the in-prompt nickname for our TGL network. The system prompt has three components. *(i)* **Role description**: the model is told it is a proactive assistant that receives the rolling session history, the current observation, and a routing list of graph-identified topics most relevant to the current activity. *(ii)* **Task formulation**: treat the first entry of the routing list as the routing anchor and use its surface text as the artifact the user is working with; quote the anchor text inline using single quotes (e.g. `'email_filter.py'`), preserving any extension or domain marker verbatim; lead with what the user *receives* (a summary, an extract, a draft, a comparison, a checklist, a function added to the anchor) and formulate the task around the supplied artifact. The task is constrained to a single sentence of at most 150 characters ending in a single period, and excludes the internal labels "TGN", "routing", "graph", or "scores". *(iii)* **Three hand-written few-shot examples**, each a JSON input bundling the session history, current observation, and routing list, paired with a JSON output. The examples cover a coding session with a quoted file anchor, a research session whose routing is intentionally empty (showing how the model uses session history when routing is empty), and a writing session with a markdown file anchor.

The *Ours* prompt is shared across all downstream backbones. *No trigger* and *Random routes* use that prompt, with the former invoking the agent at every event and the latter receiving a sampled routing list. *No TGL* retains the output-style directives and uses language-agent self-gating; its prompt omits the routing field, routing-specific instructions, and three routing demonstrations. Appendix D.1 specifies the random-list construction.

Decoding

Local vLLM backbones use temperature 0, and API backbones use the provider's default sampling settings. The output cap is 1,536 tokens. JSON output is requested through a structured-output schema where supported and through format instructions otherwise. The parsing procedure retries with `json_object` mode and then unconstrained generation, followed by regex extraction of the first JSON object. If parsing fails throughout, the raw response is sent to the benchmark judge. Each *Ours* run uses the same prompt content across backbones.

C.2 Instantiation: Mobile Agent (FingerTip)

The mobile implementation predicts one intent per FingerTip episode. It uses `Qwen2.5-VL-7B-Instruct` with t episode screenshots (t is the number of images) and graph-based retrieval candidates. The semantic head retrieves training-set intent strings, which pass through the same-app filter before entering the language agent's context. The prompt requests a Chinese intent imperative of 8–25 characters.

Candidate Retrieval

We encode each test episode with TGL and read the current-episode node's 1024-dimensional L_2-normalised semantic vector. Training vectors are precomputed for all episodes in the released training split. Cosine similarity ranks the training neighbours for each test episode. We retain the top six neighbours and remove duplicate target-intent strings before same-app filtering.

Same-App Coherence Filter

The same-app filter retains candidates whose app matches the top-ranked candidate, yielding at most two candidates. Confidence is high when the highest cosine similarity is at least 0.78 and two candidates survive, low when the highest cosine similarity is below 0, and medium otherwise. The 0.78 threshold is fixed before evaluation. The test-set confidence distribution is 56% high, 44% medium, and 0% low.

Prompt and Decoding

The user payload bundles the user profile, time, scenario, the last 20 previous intents, the retrieval-confidence label, and the retrieved intent candidates (each carrying an app, an intent class, and an intent text). When $t \geq 1$ the first t episode screenshots are loaded as JPEG bytes, base64-encoded, and prepended to the user message as image content blocks. The system prompt (in Chinese, since the target intent strings are Chinese) instructs the model to read the screenshots first, then the previous intents, then the candidates as a soft prior whose use depends on the retrieval-confidence label: high-confidence candidates may be copied as-is; medium-confidence candidates are advisory, with the predicted app grounded in the screenshots and history; low-confidence candidates are ignored entirely. The prompt requires a JSON output containing the predicted app, the predicted intent class, and a proposed-task string of 8 to 25 Chinese characters containing the predicted app name, plus session-theme and thoughts bookkeeping fields. Four hand-written few-shot demonstrations precede the test payload: three demonstrate how high-confidence candidates are combined with recent history (an alarm-setting episode anchored on 闹钟 (Clock), a video-watching episode anchored on 哔哩哔哩 (Bilibili), a food-delivery episode anchored on 美团 (Meituan)), and one demonstrates a low-confidence case in which the model synthesises its answer from the user's repeated activity in the session history (the override demonstration is anchored on 支付宝 (Alipay) drawn from the user's repeated morning routine). The override demonstration specifies the low-confidence branch of the trust-gating rule.

The local vLLM agent uses temperature 0 and a 320-token output cap. Parsing strips Markdown fences and extracts the first JSON object. The proposed-task field must contain 1–60 characters and pass a wrapper filter. This filter rejects the prefixes 用户 / 时间： / 地点： / 意图： and the pattern "用户 …意图是 …". A failed parse or filter check returns the top-ranked retrieved intent. At least 99% of outputs parse successfully under this configuration.

D Additional Experiments

D.1 Ablation Study: Full Numbers

Table 4 gives the values plotted in Figure 5, including variation across three runs. The controls address three parts of the pipeline: *No TGL* uses language-agent self-gating with the style-only prompt, *No trigger* retains learned routing and invokes the agent at every event, and *Random routes* retains the trigger and replaces the routed context. Each loss column reports how many F1 percentage points the control falls below *Ours*. Figure 5 plots absolute F1 and annotates the differences from *Ours*.

Random routing. We sample an equal-length replacement list uniformly from the distinct entity labels in the training vocabulary. The pool includes files, applications, search queries, web addresses, and category nodes. Every sampled entity is supplied with relevance score 0.5. Sampling is seeded by event, so repeated runs use the same replacement list. This control measures the effect of the supplied entity content under the deployed trigger and prompt.

Table 4: Pipeline ablations: F1 in percent (mean ± standard deviation over three runs). No TGL uses language-agent self-gating and a style-only prompt; No trigger invokes the agent at every event; Random routes replaces the entity list with equally many sampled training entities. $\Delta = \mathrm{F1}_{\mathrm{Ours}} - \mathrm{F1}_{\mathrm{control}}$ is the F1 loss in percentage points.

	No TGL		No trigger		Random routes		
Agent	F1	Loss	F1	Loss	F1	Loss	**Ours**
Local-deployable open-weight							
Qwen2-7B-Instr.	44.54±3.11	+26.1	67.95±2.59	+2.7	56.12±3.52	+14.6	**70.68±2.24**
LLaMA-3.1-8B-Instr.	49.63±2.76	+22.4	68.46±2.16	+3.6	51.38±6.09	+20.7	**72.07±3.68**
Gemma-3-12B-it	64.82±1.98	+12.5	74.05±1.22	+3.2	70.40±1.67	+6.9	**77.30±1.79**
Qwen3-4B	55.45±0.45	+11.3	62.51±2.13	+4.3	60.34±2.34	+6.4	**66.77±3.04**
Qwen3-8B	55.32±1.90	+16.8	68.85±0.90	+3.3	53.99±0.56	+18.1	**72.14±0.49**
Cloud/API							
DeepSeek-V4-Flash	67.15±1.98	+9.5	73.66±3.72	+2.9	64.94±1.46	+11.7	**76.60±3.33**
DeepSeek-V4-Pro	67.50±4.11	+6.2	71.05±4.01	+2.7	71.35±3.81	+2.4	**73.70±4.81**
GPT-4o-mini	62.80±2.11	+6.6	65.96±0.89	+3.4	63.50±1.22	+5.9	**69.38±1.33**
GPT-4o	58.17±1.67	+15.1	70.65±1.99	+2.6	63.50±1.49	+9.7	**73.23±2.64**
GPT-5.4-mini	67.14±1.90	+10.4	75.24±3.05	+2.3	69.35±2.73	+8.2	**77.55±2.40**
GPT-5.4	73.78±0.62	+2.8	74.36±2.66	+2.2	69.53±0.51	+7.0	**76.57±3.94**
Claude-Haiku-4.5	53.38±4.35	+20.0	70.47±0.88	+2.9	71.41±1.94	+2.0	**73.40±0.96**
Claude-Sonnet-4.6	64.56±2.41	+11.6	70.91±4.35	+5.3	75.14±3.08	+1.1	**76.20±3.35**
Claude-Opus-4.7	73.40±2.09	+6.5	75.86±1.52	+4.0	74.22±1.44	+5.6	**79.86±1.57**
mean	61.26	+12.7	70.71	+3.2	65.37	+8.6	**73.96**

D.2 Trigger-Architecture Baselines

Table 1 compares eight alternative triggers with TGL. The classifier rows use event labels derived from the released training set; the generation-based SFT row uses released task conversations. TGL also receives entity-routing supervision. Each trigger is evaluated with the same routing list, downstream prompt, and language-agent backbones. The inputs and optimisation settings below define the architecture comparison.

Rule. A dictionary lookup that maps each event's verb to its empirical fire rate observed in the training event-trigger labels. At inference the trigger probability is the lookup value; the trigger threshold sweeps that value.

Tabular (LR, HGB). Logistic regression (LR) and histogram gradient boosting (HGB) (`sklearn` defaults) fitted on a 65-dimensional hand-crafted feature vector per event: verb identity (one-hot), foreground-application identity (one-hot), per-entity-type counts (files, applications, search queries, web addresses, and category nodes in the current event), and time-since-previous-event in the session.

Textual (BGE-frozen + MLP, BGE-FT + MLP). A two-layer multilayer perceptron (MLP) head sits on the `[CLS]` representation of the last five session events, joined by `[SEP]` and encoded by a `BAAI/bge-base-en-v1.5` encoder. The frozen variant freezes the encoder and trains only the MLP head (0.2 M parameters), isolating the encoder's pre-trained representation; the fine-tuned (FT) variant updates the encoder end-to-end (109 M trainable). Both use the same trigger labels and positive-class weighting as TGL.

LLM-as-trigger (Qwen3-0.6B, Qwen3-8B). A chat-tuned LLM is asked one binary yes/no question per event with the rolling session history and current observation as user payload, and the trigger probability is read off the next-token logit as $P(\text{yes})/(P(\text{yes}) + P(\text{no}))$ summed over case-variant token IDs ({yes, Yes, YES} vs. {no, No, NO}). Qwen3 chat models default to reasoning mode and emit a `<think>` token first (probability ≈1 on every event without bypass); we append the empty thinking block `<think>\n\n</think>\n\n` after the chat-template prefix to expose the answer logit. Low-rank adaptation (LoRA) uses adapter rank 16, adapter scaling parameter $\alpha = 32$, dropout

0, and adapters on the attention query, key, value, and output projections, 2 epochs at lr 10^{-4} with warmup ratio 0.03 in BF16 at effective batch 32. LLM-trigger training uses a 1:1 balance of fire and no-fire events. Loss is cross-entropy on the assistant's first token only. At inference we use the merged-LoRA HuggingFace transformers eager mode (`peft.merge_and_unload()`, batch 1, `torch.cuda.synchronize()` before/after each forward) for the latency measurement; merging the adapter recovers base-model latency to within ± 3 ms.

ProactiveAgent full-SFT replication (Proactive-Qwen3-0.6B). A full-parameter supervised fine-tuning (SFT) replication of the ProactiveAgent release recipe at the 0.6 B scale: `Qwen/Qwen3-0.6B` (752 M parameters, every weight trainable, embeddings included), multi-turn conversations from the paper's released `agent_trainset.json`, AdamW at lr 10^{-5} with cosine schedule and warmup ratio 0.1, effective batch 32 (per-device $1\times$ grad-accum 32, matching the paper's 8 GPU $\times$ per-device $1\times$ grad-accum 4), 10 epochs, sequence length 4,096, BF16, max grad norm 1.0, assistant-tokens-only cross-entropy loss. Wall-clock 520 min on a single NVIDIA RTX A6000 (48 GB). At inference the per-event protocol matches the released-checkpoint protocol of Lu et al. (2025): the system prompt is the paper's Role/Task/Format/Rules block, the per-event user payload is the rolling session history plus the current observation as a JSON `{Instructions, Observation}` object, and the trigger is derived from the generated `proposed_task` field (non-empty $\Rightarrow$ prob 1.0, empty $\Rightarrow$ 0.0), averaged over 3 inference seeds. Sampling follows the Qwen3 non-thinking defaults (sampling temperature 0.7, nucleus probability 0.8, top-20 token sampling, and a 512-token cap; mean 139 generated tokens per event). This row is the only LLM trigger in Table 1 that derives its decision from a full task-text generation; its Latency columns therefore measure decoding cost.

Score distributions. Figure 6 shows the per-event trigger scores of the eight single-forward architectures. It describes their score scales alongside the ranking and calibration metrics in Table 1.

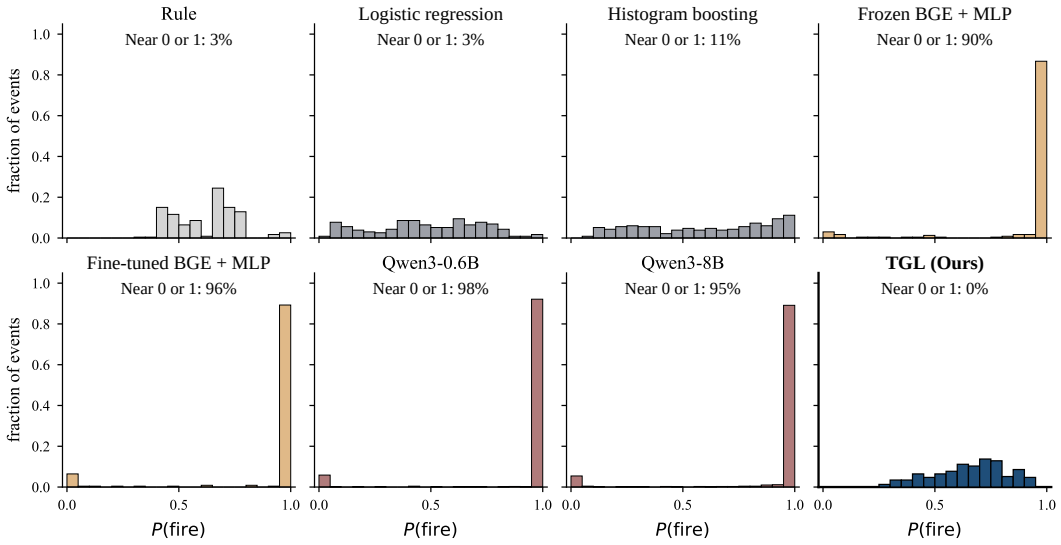


Figure 6: Distribution of trigger probabilities over desktop test events for eight models evaluated with one forward pass. Horizontal axes show the predicted probability of firing; vertical axes show the fraction of events in each bin of width 0.05. Panel annotations give the percentage below 0.05 or at least 0.95. LR: logistic regression; HGB: histogram gradient boosting; MLP: multilayer perceptron. Fixed-threshold F1 is evaluated in Appendix E.2.

D.3 Drop-In TGL Trigger on Fine-Tuned Downstream Agents

We also examine agents already fine-tuned for proactive assistance, testing whether graph-based filtering improves intervention selection while retaining their existing suggestions. We apply it offline to the two fine-tuned baselines from Lu et al. (2025), LLaMA-3.1-8B-Proactive and Qwen2-7B-Proactive: events whose trigger probability falls below the deployed threshold are forced to `task = null`; everything else (their published outputs, prompts, and decoding) is unchanged.

Figure 7 plots F1 versus the applied trigger threshold, with a dashed line at the model's published F1 from Table 5 (66.25 for LLaMA-3.1-8B-Proactive, 66.47 for Qwen2-7B-Proactive). At our deployed threshold, both checkpoints exceed their published F1: LLaMA-3.1-8B-Proactive reaches 67.66 (+1.4 F1) with 25% fewer LLM calls, and Qwen2-7B-Proactive reaches 67.38 (+0.9 F1) with 21% fewer LLM calls. The sweep shows how the trade-off changes with the threshold: stronger filtering eventually removes useful interventions and lowers F1.

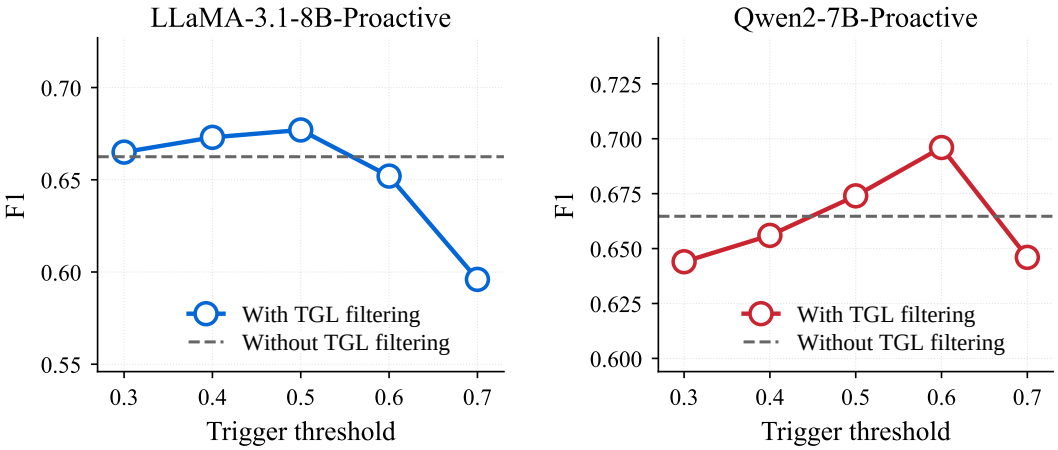


Figure 7: Filtering the outputs of two fine-tuned proactive agents. Coloured curves show F1 after applying the TGL trigger at each threshold; grey dashed lines show the same fine-tuned agents without filtering. The deployed threshold is 0.5. Outputs retained by the filter keep their original suggestions.

D.4 Full Per-Backbone Main Results

Table 5 gives every metric behind Figure 4: precision, recall, accuracy, false-alarm rate, and F1 for each of the 14 backbones under the vanilla protocol and with TGL triggering and routing, plus the two fine-tuned proactive baselines.

D.5 Per-Backbone Downstream-Agent Latency

To quantify the generation cost avoided by skipping an event, Table 6 reports the wall-clock cost of one downstream-agent call for each of the 14 backbones, observed end-to-end under our evaluation serving setup (open-weight on an A6000 vLLM; OpenAI / Anthropic / DeepSeek over their respective official APIs). They include API queueing and load, which is the per-event cost a deployed system pays. The fastest measured API call (1503 ms for GPT-5.4-mini) is $\approx 135\times$ TGL's trigger forward.

E Additional Robustness and Calibration Analyses

We evaluate backward temporal edges, fixed-threshold performance, graph-construction choices, and events with an empty routing list. Retraining follows Appendix F; downstream evaluations use five local open-weight backbones with three runs each. For the graph-structure analyses, trigger AUC uses the must-fire target from Section 5.4, with 0.738 for the deployed model. The must-fire label marks events requiring a non-null task in the released test annotations. Routing-anchor agreement is the fraction of events with extracted entities on which a variant selects the same highest-scoring entity as the deployed model; retraining the original recipe gives agreement of 0.90.

E.1 Backward-Temporal-Edge Ablation

The training graph includes forward and backward temporal edges. The deployed inference procedure removes backward temporal edges and masks future-node input representations (Appendix F.7). We retrain with both edge configurations to examine how backward training edges affect triggering and routing under the deployed inference procedure. A separate variant adds backward edges at inference.

Table 5: Full desktop results in percent. Each backbone is evaluated with the benchmark's vanilla prompt and with TGL triggering and routing (indented row). Bold marks the better value in each pair. Fine-tuned baselines and the first four vanilla rows are published results from Lu et al. (2025); the remaining vanilla rows are our re-runs with the same reward-model judge. Our runs average three evaluations (Appendix F.1). False alarm is the fraction of generated suggestions rejected by the judge.

Model	Precision ↑	Recall ↑	Accuracy ↑	False alarm ↓	F1 ↑
Fine-tuned proactive baselines					
LLaMA-3.1-8B-Proact.	49.76	99.06	52.86	50.24	66.25
Qwen2-7B-Proact.	49.78	100.00	50.66	50.22	66.47
Backbones with published vanilla results					
Qwen2-7B-Instr.	44.00	**98.02**	43.61	56.00	60.74
With TGL	**55.56**	97.22	**62.66**	**44.44**	**70.68**
LLaMA-3.1-8B-Instr.	38.16	**98.86**	39.06	61.84	55.06
With TGL	**57.32**	97.29	**64.09**	**42.68**	**72.07**
GPT-4o	48.15	**98.11**	49.78	51.85	64.60
With TGL	**58.73**	97.36	**65.24**	**41.27**	**73.23**
GPT-4o-mini	35.28	**100.00**	36.12	64.73	52.15
With TGL	**53.97**	97.14	**61.37**	**46.03**	**69.38**
Backbones with re-run vanilla results					
Gemma-3-12B-it	42.67	91.05	46.07	57.33	58.04
With TGL	**64.02**	**97.58**	**69.53**	**35.98**	**77.30**
Qwen3-4B	23.89	**100.00**	25.75	76.11	38.54
With TGL	**50.97**	96.97	**58.94**	**49.03**	**66.77**
Qwen3-8B	15.07	**99.07**	19.17	84.93	26.14
With TGL	**57.32**	97.31	**64.09**	**42.68**	**72.14**
GPT-5.4-mini	52.97	94.74	63.23	47.03	67.94
With TGL	**64.37**	**97.59**	**69.81**	**35.63**	**77.55**
GPT-5.4	58.88	**97.72**	64.38	41.12	73.47
With TGL	**63.14**	97.54	**68.81**	**36.86**	**76.57**
Claude-Haiku-4.5	45.03	**99.31**	50.93	54.97	61.89
With TGL	**58.91**	97.38	**65.38**	**41.09**	**73.40**
Claude-Sonnet-4.6	49.87	71.23	58.85	50.13	58.64
With TGL	**62.61**	**97.52**	**68.38**	**37.39**	**76.20**
Claude-Opus-4.7	65.71	85.86	**72.53**	34.29	74.44
With TGL	**67.55**	**97.70**	72.39	**32.45**	**79.86**
DeepSeek-V4-Flash	51.25	83.73	57.94	48.75	63.44
With TGL	**63.14**	**97.54**	**68.81**	**36.86**	**76.60**
DeepSeek-V4-Pro	43.26	52.37	58.94	56.74	47.12
With TGL	**59.44**	**97.38**	**65.81**	**40.56**	**73.70**

Table 6: Wall-clock latency of one downstream generation call and its ratio to the 11.13 ms TGL controller latency. Measurements include request handling and API queueing. *Serving:* open-weight backbones via A6000 vLLM; OpenAI / Anthropic / DeepSeek backbones via the respective vendors' official APIs.

Backbone	Latency (ms/call)	Ratio to controller
Open-weight (A6000 vLLM)		
Qwen2-7B-Instr.	2412	217×
LLaMA-3.1-8B-Instr.	2638	237×
Gemma-3-12B-it	4461	401×
Qwen3-4B	1800	162×
Qwen3-8B	2634	237×
Closed/API		
DeepSeek-V4-Flash	5705	513×
DeepSeek-V4-Pro	21839	1962×
GPT-4o-mini	2387	215×
GPT-4o	1688	152×
GPT-5.4-mini	1503	135×
GPT-5.4	2431	218×
Claude-Haiku-4.5	4366	392×
Claude-Sonnet-4.6	5714	513×
Claude-Opus-4.7	5676	510×

Setup. We retrain in two arms: (a) *both-direction training*, using earlier-to-later and later-to-earlier event edges; (b) *forward-only training*, using earlier-to-later event edges. We use five training seeds per condition and evaluate both with the deployed inference procedure.

Trigger AUC. Table 7 reports mean trigger AUC of 0.733 for both training arms. The paired per-seed difference is 0.000 ± 0.011. The deployed checkpoint's AUC of 0.738 lies within the retraining range of 0.722–0.744.

Table 7: Effect of temporal-edge direction during training. Trigger AUC uses the released must-fire labels. Retrained rows report mean ± standard deviation over five seeds; both use the deployed inference procedure.

Training configuration	Must-fire AUC ↑
Deployed model (Table 1)	0.738
Both temporal directions	0.733 ± 0.009
Forward temporal edges only	0.733 ± 0.008
Paired difference (forward-only − both)	0.000 ± 0.011

Routing-anchor agreement. The two training arms select the same top-ranked routing entity on 96% of anchored events, measured with paired seeds. Retraining the original recipe yields 90% reference agreement with the deployed model.

End-to-end F1. Each arm contributes one retrained checkpoint to the full pipeline, evaluated on five open-weight backbones with three downstream runs each. Table 8 reports differences between the run means: removing backward edges changes mean F1 by -0.5 points. The two arms reach mean F1 of 71.9 and 71.5, compared with 71.8 for the deployed checkpoint on these five backbones.

Inference-edge variant. Adding backward temporal edges at inference changes the deployed checkpoint's trigger AUC by less than 0.005, preserves the routing selections on every test event, and changes end-to-end F1 by $+0.03 \pm 0.3$ across five backbones and three runs.

Table 8: Effect of removing backward training edges on downstream F1. Each condition uses one retrained model with three runs per backbone. Δ is forward-only training minus both-direction training, in F1 percentage points.

Downstream backbone	Δ F1 (percentage points)
Qwen2-7B-Instr	-0.2
Qwen3-4B	$+1.3$
Qwen3-8B	$+2.3$
LLaMA-3.1-8B-Instr	-4.1
Gemma-3-12B-it	-1.7
mean	$\mathbf{-0.5}$

E.2 Calibration and Fixed-Threshold Reliability

Probability calibration. We evaluate whether the trigger's numerical probabilities match the frequency with which assistance is needed. For this analysis, the released reward-model judge evaluates an empty suggestion at every event: rejection of that suggestion gives label 1 (fire), and acceptance gives label 0 (skip). We fit the calibrator on judge-labelled training events and evaluate on judge-labelled test events, with positive rates of 55.25% and approximately 34%, respectively. These rates use all judged events in the corresponding split as their denominator. The original trigger-training labels come from proposed tasks and have a positive rate of 88% (Appendix B.1).

Temperature scaling adjusts the trigger's probability scale after training. Given its logit z, the calibrated probability is $p = \sigma(z/T)$, with fitted $T \approx 3.27$. The positive temperature preserves score ordering and decisions at $\tau = 0.5$. The Brier score is the mean squared error between probability and binary label. Expected calibration error (ECE) averages the absolute gap between predicted probability and observed positive rate over ten equally spaced probability bins, weighted by the number of events in each bin. Table 9 uses the judge-labelled fire/skip target for all three metrics: scaling reduces Brier from 0.310 to 0.259 and ECE from 0.336 to 0.227, while AUC remains 0.703.

Table 9: Post-training calibration of trigger probabilities. Temperature scaling uses $p = \sigma(z/T)$, with positive T fitted on judge-labelled training events. All metrics use the same judge-labelled fire/skip target on the test split: a positive label means the judge rejects an empty suggestion. The final column reports the percentage of test events whose decisions change after calibration at $\tau = 0.5$.

Probability mapping	AUC ↑	Brier ↓	ECE ↓	Changed decisions
Uncalibrated	0.703	0.310	0.336	–
Temperature-scaled	0.703	0.259	0.227	0%

Threshold sensitivity. We vary the threshold around the deployed $\tau = 0.5$ to examine how sensitive suggestion quality is to the operating point. Across five open-weight backbones with three runs each, mean F1 ranges from 70.4 to 73.1 over $\tau \in [0.4, 0.6]$, with 72.1 at the deployed threshold. The wider sweep $\tau \in [0.3, 0.7]$ yields 67.8–73.5. Post-hoc per-run threshold selection on the test set improves mean F1 by at most 1.5 points over the deployed threshold.

Cross-backbone threshold transfer. To assess threshold reuse when changing the language agent, we choose the threshold maximising mean F1 on four open-weight backbones and apply it to the held-out fifth, rotating through all five. This analysis uses the same benchmark events across backbones. The transferred threshold incurs a mean loss of 0.4 F1 relative to each held-out backbone's own tuned threshold (Table 10).

E.3 Construction-Rule Sensitivity

Graph construction determines both the routing candidates and message-passing context. We vary entity categories and feature channels to test sensitivity to these choices; random entity deletion tests sensitivity to missing context. Category nodes encode file extension, file category, website domain, or query language (Appendix B.1). Each node-removal variant removes the named nodes and their

Table 10: Threshold transfer across five backbones on the same test events. A threshold is selected using four backbones and applied to the fifth. Δ F1 is transferred-threshold F1 minus the held-out backbone's individually tuned F1, in percentage points; each mean uses three downstream runs.

Held-out backbone	Δ F1 (percentage points)
Qwen2-7B	-0.2
Qwen3-4B	-0.5
Qwen3-8B	-0.1
LLaMA-3.1-8B	-1.1
Gemma-3-12B	-0.2
mean	$\mathbf{-0.4}$

incident edges; each feature variant disables the named embedding. Random deletion removes 25% of entities. We retrain the controller for each variant. The variants use three training seeds and the deployed inference procedure; the reference and entity-deletion rows reuse the existing retraining runs. Table 11 reports trigger AUC for decision quality and anchor agreement for changes in entity selection.

Table 11: Sensitivity to graph construction. AUC measures trigger ranking against must-fire labels (mean $\pm$ standard deviation across training seeds). Routing agreement is the fraction of events with entities where the highest-ranked entity matches the deployed model. All rows use the deployed inference procedure.

Construction change	Must-fire AUC $\uparrow$	Routing agreement
Original construction	0.733 ± 0.009	0.90
Remove all category nodes	0.717 ± 0.012	0.78
Remove search-query and language-category nodes	0.744 ± 0.008	0.86
Remove web-address and domain nodes	0.735 ± 0.005	0.89
Disable node-type embedding	0.725 ± 0.004	0.83
Disable time-gap embedding	0.715 ± 0.016	0.89
Randomly remove 25% of entities	0.725 ± 0.019	0.71

Trigger prediction. Mean AUC ranges from 0.715 to 0.744 across the construction variants, compared with 0.733 ± 0.009 for the reference. Disabling the time-gap embedding gives 0.715 ± 0.016, removing category nodes gives 0.717 ± 0.012, and random deletion of 25% of entities gives 0.725 ± 0.019.

Routing selections. Anchor agreement ranges from 0.83 to 0.89 when individual entity classes or feature channels are removed. Removing category nodes gives agreement of 0.78, and random entity deletion gives 0.71. These measurements describe the routing selections under each change to the graph's candidate entities and features.

E.4 Events with No Extracted Entities

The controller processes every observation, including those with no extracted entities to route. We therefore examine trigger ranking and suggestion quality separately for events with and without extracted entities. At 34% of test events, the current observation yields an empty entity set and the routing list is empty. For example, *"The user scrolls and clicks on a research report page about biodegradable cups."* contributes its text and temporal position to the session graph. Table 12 reports both groups for five open-weight backbones with three downstream runs each.

Triggering from event text and history. The event readout aggregates its text, timing, and neighbouring session context. Trigger AUC is 0.789 on events with no extracted entities and 0.723 on events with extracted entities, with fire rates of 59% and 92%. Within one session, the report-scrolling event above fires with $p = 0.73$ and carries a must-fire label. A later housekeeping event, *"The user types 'cmdw' to close a tab and continues with scrolls and clicks."*, receives $p = 0.27$ and is skipped.

Table 12: Performance by entity availability in the current observation. Event share uses all test events; must-fire rate is the positive-label fraction within each group. Predicted fire rate uses $\tau = 0.5$. End-to-end F1 reports mean $\pm$ standard deviation across five backbones with three runs each; full event text and history remain available to both groups.

Current observation	share of events	must-fire rate	Trigger AUC (must-fire)	Predicted fire rate at τ=0.5	end-to-end F1
No entities extracted	34%	0.19	**0.789**	59%	**75.0 $\pm$ 6.8**
Entities extracted	66%	0.26	0.723	92%	71.0 $\pm$ 4.4

Generating suggestions from session history. For an empty routing list, the downstream prompt uses the session history to formulate the task (Appendix G). Mean end-to-end F1 is 75.0 on events with no extracted entities and 71.0 on events with extracted entities.

F Implementation Details

This appendix lists the architectural and training details deferred from Section 4 and Section 5.

F.1 Evaluation Backbones

Fourteen instruction-following backbones serve as the downstream language agent in the panel evaluations of Table 5 and the ablations of Tables 4 and 1:

- *Open-weight (5)*, served via vLLM on a single A6000: Qwen2-7B-Instruct, LLaMA-3.1-8B-Instruct, Gemma-3-12B-it, Qwen3-4B, Qwen3-8B.
- *OpenAI (4)*, served via the OpenAI API: GPT-4o-mini, GPT-4o, GPT-5.4-mini, GPT-5.4.
- *Anthropic (3)*, served via the Anthropic API: Claude-Haiku-4.5, Claude-Sonnet-4.6, Claude-Opus-4.7.
- *DeepSeek (2)*, served via the DeepSeek API: DeepSeek-V4-Flash, DeepSeek-V4-Pro.

Two fine-tuned proactive baselines from Lu et al. (2025) (LLaMA-3.1-8B-Proactive and Qwen2-7B-Proactive) appear for reference in Table 5 and Figure 7; both are open-weight checkpoints with published reference results.

RM judge. All ProactiveAgent precision, recall, accuracy, false-alarm, and F1 numbers are computed with the released reward-model judge from Lu et al. (2025). The originating paper trains this RM on human-annotated event–prediction judgments and reports above-90 agreement with held-out human annotations in its reward-model assessment (91.80 F1 and 91.67 accuracy).

Run aggregation. Our downstream evaluation cells aggregate n=3 runs: provider-default sampling for API backbones and temperature 0 for local vLLM backbones (run seeds $\{0, 1, 100\}$). Published legacy vanilla and fine-tuned reference values retain the originating paper's aggregation, as marked in Table 5. Each trigger-architecture metric in Table 1 (AUC, Brier, ECE, threshold standard deviation) is likewise averaged over 3 inference runs (seeds $\{0, 1, 100\}$) of the same trained checkpoint; the trained trigger checkpoint is fixed across these inference runs.

F.2 Node Featurisation

Desktop nodes have three input features. *(i)* **Text embedding:** a 768-dimensional vector from the frozen `BAAI/bge-base-en-v1.5` encoder. Event nodes use the full event text, named entities use their surface labels, and category nodes use a serialised type–value label. For example, the Python-extension category is encoded as `file_ext:py`. This literal string specifies the encoder input. *(ii)* **Node-type embedding:** a learned 32-dimensional vector identifying the node as an event, application, file, web address, search query, document artifact, file extension, file category, query-language category, or website domain. *(iii)* **Time-gap embedding:** a 32-dimensional multilayer perceptron (MLP) representation of $\log(1+s)$, where s is the number of seconds since the preceding event. The first event and entity nodes use $s = 0$.

Table 13: Resources used by the trigger models in Table 1. Total parameters include frozen encoders; training time is wall-clock time. LR: logistic regression; HGB: histogram gradient boosting; MLP: multilayer perceptron; FT: fine-tuning; SFT: supervised fine-tuning. Tree models have no neural parameter count.

Trigger	Total parameters	Trainable	Training time
Rule	0	0	<1 s
LR (65 features)	65	65	41 s
HGB (65 features)	–	–	4 s
BGE-frozen + MLP	109.5 M	0.2 M	$\sim$5 s
BGE-FT + MLP	109.5 M	109.5 M	23 min
Qwen3-0.6B	757 M	4.59 M	19 min
Qwen3-8B	8.21 B	15.34 M	86 min
Proactive-Qwen3-0.6B (full SFT)	752 M	752 M	520 min
TGL (Ours)	**110.4 M**	**1.16 M**	**$\sim$5 min**

The frozen text encoder runs once per minibatch. Concatenating the three features gives an 832-dimensional vector, which a linear projection maps to the 256-dimensional graph representation. Appendix B.2 specifies the mobile node vocabulary and Chinese text encoder.

F.3 GNN Body

Message passing runs for three rounds of relation-aware GATv2 attention (Brody et al., 2022) (implemented as `torch_geometric.nn.GATv2Conv`), each with: *(i)* 4 attention heads, head dimension $256/4 = 64$, concatenated; *(ii)* a 32-dimensional edge embedding (one entry per edge relation type) fed into each layer's attention; *(iii)* explicit `self` edges with learned relation embeddings, supplied to GATv2Conv with `add_self_loops=False`; and *(iv)* a residual + LayerNorm + GELU + dropout 0.2 block around each layer, $x \leftarrow \mathrm{LN}(x + \mathrm{Drop}(\mathrm{GELU}(\mathrm{GATv2}(x, e, \text{edge_emb}))))$.

F.4 Combining Graph Layers

Following Xu et al. (2018), we keep the output of the input projection plus every GNN layer:

$$h_{\text{jk}} = \text{concat}(\text{layer}_0, \text{layer}_1, \text{layer}_2, \text{layer}_3),$$

of dimension $256 \times 4 = 1024$. Both heads receive the node's input features and the context aggregated at each graph layer.

F.5 Heads

Two parallel MLPs, each `Linear`$(1024 \to 256) \to$ `GELU` $\to$ `Dropout`$(0.2) \to$ `Linear`$(256 \to 1)$: the **trigger head** is applied only to event nodes and outputs the trigger logit; the **routing head** is applied only to entity nodes and outputs the per-entity relevance logit. Total trainable parameters: 1.16 M.

F.6 Training

Each batch produces one trigger logit per event and one routing logit per entity. We mask both outputs to nodes that have a label and use binary cross-entropy (BCE) losses computed from the logits:

$$\begin{aligned} \mathcal{L}_{\text{trig}} &= \mathrm{BCE}(\text{logit}_{\text{trig}}, y_{\text{trig}}; w_+ = 0.15), \\ \mathcal{L}_{\text{rout}} &= \mathrm{BCE}(\text{logit}_{\text{rout}}, y_{\text{rout}}), \\ \mathcal{L} &= 0.3\,\mathcal{L}_{\text{trig}} + 1.0\,\mathcal{L}_{\text{rout}}. \end{aligned}$$

Notes. A label of -1 flags an unlabelled node and is excluded from the loss. The trigger positive-class weight $w_+ = 0.15$ down-weights the dominant positive class in the training set. The relative loss weighting in Eq. (5) is fixed at $\lambda_{\text{trig}} = 0.3$ and $\lambda_{\text{rout}} = 1.0$; the hidden width is $d = 256$ with $K = 4$ attention heads.

Optimisation. AdamW with learning rate 3×10^{-4}, weight decay 10^{-4}, gradient clipping at global norm 1.0. Each gradient step batches 32 session graphs as disjoint graph components. We use the epoch-2 checkpoint with the fixed inference settings in Appendix F.8.

Hardware and runtime. All measurements run on a host with 8 NVIDIA RTX A6000 GPUs (48 GB each), $2\times$ Intel Xeon Gold 6254 CPUs (72 threads at 3.10 GHz), and 251 GiB RAM. Training and per-event latency benchmarks use a single A6000. One epoch over the training set is ≈ 140 s wall-time, dominated by the BGE forward; training to 2 epochs takes about 5 minutes total.

F.7 Inference on Event Histories

Training encodes complete sessions with forward and backward temporal edges. For each event, we evaluate the session graph with backward temporal edges removed and future-node input representations masked. Trigger scores are read at e_t, and routing scores are read for entities connected to e_t. Appendix E.1 evaluates the backward-edge channel under the deployed inference procedure.

F.8 Inference Hyperparameters

Trigger threshold $\tau_{\text{trig}} = 0.5$ (the sigmoid decision boundary); routed-list cap $k = 20$. These hyperparameters are held fixed across all backbones and ablation cells.

F.9 Latency Measurement Protocol

The Latency columns of Table 1 are measured under two hardware regimes. *Server* runs on a single NVIDIA A6000; the Rule, LR, and HGB rows run on the host's dual Intel Xeon Gold 6254 CPUs. *Local* runs on an Apple M3 Pro laptop with 36 GB unified memory: Textual, LLM, and Graph rows execute under Metal Performance Shaders (MPS); Rule, LR, and HGB run on the M3 Pro CPU. Transformer inference uses auto padding under streaming-serve protocol (events arrive one at a time, the encoder is called per event), with the same dynamic-padding policy across transformer rows. Each cell is the mean of the last 3 of 4 timed forward passes; the first pass is discarded to absorb MPS compilation. All rows use eager inference at batch size one; the TGL row uses the cached streaming protocol in Section F.10, reusing recurring entity embeddings while encoding the event text and fresh entity strings per event. The Qwen3 LoRA rows on Local report the mean across the same three inference seeds, identical recipe to the server-side measurement. The Proactive-Qwen3-0.6B (full SFT) row is the one exception to single-forward measurement: deriving its binary trigger requires generating the full proposed-task text and then thresholding on whether the generation is non-empty, so each event triggers a full decoding pass (mean 139 generated tokens; sampling temperature 0.7, nucleus probability 0.8, top-20 token sampling, and a 512-token cap). Its latency cells therefore report full decoding cost: $3{,}927$ ms on server and $11{,}966$ ms on local, averaged across 3 inference seeds.

F.10 Cached Streaming Serving

The latency reported in Table 1 for TGL (11.13 ms server, 13.99 ms local) uses cached streaming serving under HuggingFace transformers + PyG inference: recurring entity embeddings are reused, while the event text and fresh entity strings are encoded per event. The measurement uses batch-1 streaming inference with no compilation, quantization, or cross-event batching.

The graph representation supports entity-level embedding caching during streaming inference. The BGE encoder is the dominant forward cost ($\sim$7–8 of the 11.13 ms), and most entity nodes a typical session touches are drawn from a small recurring vocabulary (applications, file extensions, file categories, query-language categories, website domains). Caching these embeddings reduces per-event BGE work to encoding only the event text plus any fresh entity strings (specific filenames, search queries, web addresses).

G System Prompts

This appendix reproduces the system prompts used by the downstream agent on each benchmark. Appendix C describes the input and output formats, few-shot settings, and decod-

Table 14: Text-embedding computation during streaming inference. Known application and category labels can be encoded before streaming. New event text and unseen entity strings are encoded when encountered; recurring entity strings reuse their cached embeddings.

Node text	When the embedding is computed
Event description	At each incoming event
File name	At first appearance; reused on recurrence
Web address	At first appearance; reused on recurrence
Search query	At first appearance; reused on recurrence
Application name	Precomputed for known applications
File extension	Precomputed for the extension inventory
File category	Precomputed for the five file categories
Query-language category	Precomputed for the four script categories
Website domain	Precomputed for known deployment domains

ing. The literal prompts retain the implementation nickname TGN for TGL. In desktop inputs, `session_history` contains recent event texts, `current_observation` contains the current event, and `tgn_routing_nodes` contains the ranked entity list; each entry has its surface label (`node`) and relevance score (`score`). In mobile inputs, `tgn_candidates` contains retrieved training intents, and `tgn_confidence` is the high/medium/low retrieval-confidence label defined in Appendix C.2. The output field `task` is the suggestion submitted for evaluation.

G.1 Desktop ProactiveAgent

System prompt

```
You are a proactive assistant. You are shown:
1. session_history: the last few user activities in this session (oldest first)
2. current_observation: what the user is doing right now
3. tgn_routing_nodes: graph-identified topics most relevant to the current activity

Your task: infer the user's current TASK THEME from the session history,
then generate ONE short, specific, actionable suggestion grounded in that theme.

Rules:
- Use session_history to identify what the user is working on (the session theme).
- If tgn_routing_nodes contains specific items (files, topics), anchor your task
  to one of them.
- The task must be a single sentence, concrete, max 150 characters.
- Reference the session theme explicitly (e.g., the programming language,
  document topic, tool).
- Do NOT give a generic suggestion like 'open the last file' or 'check recent
  activity'.
- Do NOT mention TGN, routing, graph, or scores.

TGN-grounded task formulation:
- The first entry of tgn_routing_nodes is the routing anchor: the entity the
  graph identified as the user's current focus. Build the task around delivering
  something concrete about that anchor; quote its surface text inline
  (e.g. 'email_filter.py') and preserve any file extension or domain marker
  verbatim.
- Lead with what the user RECEIVES, not what they should DO next. Because the
  routing anchor has already located the entity, do not mirror the user's last
  observed verb (open / search / browse) back as the task. Instead state the
  deliverable: a summary, an extract, a draft, a comparison, a checklist, a
  function added to the anchor.
- End the task with a single period. No question mark, no exclamation, no
  trailing offer or hedge: the routing score already encodes confidence, the
  surface text should match.

Output ONLY JSON: {"task":"string","session_theme":"string","purpose":"string",
"thoughts":"string","response":"string","intent":"string"}.
```

No-TGL Prompt Variant (Style Only)

The *No TGL* ablation in Table 4 replaces the production system prompt with the variant below. It retains the style directives (single sentence, 150-character cap, lead-with-deliverable, do-not-invent-artifacts) and removes every reference to `tgn_routing_nodes`, the routing-anchor instructions,

and all three few-shot demonstrations. The trigger is disabled; the agent self-gates by emitting `task = ""`.

```
You are a proactive assistant. You are shown:
1. session_history: the last few user activities in this session, oldest first.
2. current_observation: what the user is doing right now.

Your task: infer the user's current TASK THEME from the session history,
then generate ONE short, specific, actionable suggestion grounded in that theme.

Rules:
- Use session_history to identify what the user is working on.
- The task must be a single sentence, concrete, max 150 characters.
- Reference the session theme explicitly.
- Do NOT give a generic suggestion like "open the last file" or "check recent activity".
- Do NOT invent files, URLs, pages, or topics not present in the input.
- Lead with what the user RECEIVES, not what they should DO next.
- End non-empty tasks with a single period.
- If the current moment does not support a useful, specific suggestion, set task to "".

Output ONLY JSON: {"task":"string","session_theme":"string","purpose":"string",
"thoughts":"string","response":"string","intent":"string"}.
```

G.2 FingerTip Mobile Proactive Agent

The FingerTip prompt requests a Chinese intent string. Each candidate supplies an application, an intent class, and an intent description; the confidence label tells the agent how to combine candidates with screenshots and history.

System prompt

```
你是一个安卓助手，根据用户画像、时间、场景、最近意图、(如果有) 当前屏幕截图，
以及 TGN 时序图谱召回的候选意图，预测用户当前一句简短的中文意图。

输入字段说明:
- previous_intents: 该用户最近的意图 (按时间升序)。是用户长期偏好的强信号。
- screenshot: 当前打开的 App 与界面，是 ** 最强证据 ** (如果有)。
- tgn_candidates: 图谱召回的候选意图 (按相关度排序)。** 软先验，可信度由
  tgn_confidence 给出 **:
  - high: 候选可信，强烈建议参考；可能直接 copy 一句作为 task。
  - medium: 候选有用但需谨慎；选 App 应跟截图/历史一致。
  - low: 候选不可信，** 忽略 tgn_candidates**，只看截图与历史。
- 候选 App 跟截图/历史所示 App 不一致时，** 永远以截图/历史为准 **，可以独立生成意图。

输出严格 JSON:
{
  "session_theme": "20 字内总结用户最近在做什么 (例: '早晨重复设闹钟'、'晚上看哔哩哔哩')",
  "thoughts": "20 字内说明你怎么定下 task 的",
  "app": " 用户当前使用的 App 中文名 (要跟截图/历史/候选三者最强证据一致) ",
  "intent_class": " 用户行为大类 (如 闹钟/计时、视频、商品搜索与购物、信息查询...) ",
  "task": " 一句 8-25 个汉字的中文祈使句，以动词或 App 名开头，包含 app 字段中的
          App 名，禁止 用户/时间: /地点: /意图: 前缀，禁止日期/星期/画像信息"
}

只输出 JSON，不要 markdown 包裹、不要前后说明。
```

H Case Study

To inspect how routed context enters the generated task, we compare suggestions for the same observation. We pick three test events where all three pipelines (the benchmark's vanilla language agent, *Ours*, *Random routes*) fire and produce *different* tasks. Each case shows the routing context the downstream agent received under *Ours*, and the reward-model (RM) verdict for every variant. The downstream agent is Qwen2-7B-Instruct under the deployed configuration; *Random routes* replaces the routing list (up to 20 entries) with draws from the training entity pool (Section 5.3).

Case 1: coding session, romanised-Chinese query. TGL routes to the query-language category labelled English and the search query *dianz youjian dizhi jiaoyan4 python*. The language category follows the Latin-script heuristic defined in Appendix B.1. Vanilla emits "*Translate the message from Chinese to English.*" (rejected). *Ours* quotes the query verbatim and ties it to the working file: "*Translate 'dianz youjian dizhi jiaoyan4 python' into English for context in the 'aw_tools.py' file.*"

(accepted). *Random routes* proposes a summary of the file from session history: "*Extract key points from the 'aw_tools.py' file and compile them into a concise summary.*" (rejected). The accepted suggestion explicitly names both the routed query and the file from session history.

Case 2: writing session, file and category context. TGL's top-ranked routing entries for this event include the file `brandResearch.md` along with its writing-file category, Markdown-extension category, and the Visual Studio Code application. *Ours* grounds on the file: "*Review and refine the structure of 'brandResearch.md' for better readability.*" (accepted). Vanilla proposes a research task: "*Search for key metrics and indicators used in measuring the sustainability of brands and provide them to the user for reference.*" (rejected). *Random routes* uses the file from session history and proposes an audience-engagement section: "*Add a section on audience engagement techniques to the 'brandResearch.md' file.*" (rejected).

Case 3: application-only routing. TGL routes to `[VSCode]` only and defers the artifact to session history. *Ours* produces a deliverable aligned with the user's note-taking activity: "*Extract and summarize Katla's product pricing details for your Markdown notes.*" (accepted). Vanilla proposes a product comparison: "*Offer to compare the pricing of Katla's products with similar items from other eco-friendly brands, providing insights on positioning.*" (rejected). *Random routes* proposes a comparison table: "*Add a table comparing prices of Katla's eco-friendly products in your Markdown notes.*" (rejected). The generated suggestion refers to the pricing information and Markdown notes present in the session context.

The three cases show how the downstream agent combines routed entities with session history. Each variant's quoted output and RM verdict are shown together, allowing the task formulation to be compared on the same observation.

I Discussion

Graph structure and continuous control. Events, recurring entities, and temporal connections provide the controller's representation. Shared message passing supplies an event-level trigger and entity-level routing scores in one forward pass. The two heads train jointly, and their outputs form a common interface for downstream agents. The routing-content ablation measures the interface's effect: the learned routing lists improve mean F1 by 8.6 points over equal-length random lists under the same trigger and prompt.

Resource and latency implications. The controller runs at 11.13 ms per event on an A6000 and 13.99 ms on an M3 Pro. Its runtime combines BGE text encoding with message passing over the session graph. Recurring entity embeddings are cached during streaming inference. The $\sim$220 MiB BF16 resident footprint supports local activity processing, and triggered events receive downstream language-agent generation. Section 5.5 details monitoring and downstream generation costs.

Compatibility with proactive checkpoints. The trigger also serves as a pre-filter for existing proactive language agents. Appendix D.3 applies it to the published outputs of LLaMA-3.1-8B-Proactive and Qwen2-7B-Proactive. At the deployed threshold, F1 increases by 1.4 and 0.9 points, respectively, and the corresponding call reductions are 25% and 21%. The retained events use the original checkpoints' suggestions.