IntentSpider - The Fluid Language Web for Textual Tension and Prediction

Neksha V. DeSilva

FIA Operating Systems, Colombo, LK
n.desilva@fiaos.org

Abstract. IntentSpider is a next word prediction engine created according to a human's function typed personal intent graph. It preserves connections through a transmission fidelity (TF) value changed by prediction results, while creating word history or context through a fan dispersion algorithm, ranking words by positive diffusion value. A block and release process executes after ranking, and a read only 2D data output depicts the saved graph state. Based on C++ version 17, it powers an open source research playground with next word prediction, suggestion insertion and a built in terminal (WebAssembly). Experiment data includes multiple updates, 10 fixed state replay measurements, 4 keyboard comparisons, runtime measurements and two separate display state systems. Across approx. 1000 continuous next word events, the first rank and top 3 exact target word accuracies are 13.2 and 25.4% at a 2.4 ms median latency, and the most recent word in the input seed produces the greatest ranking related difference with a calculated 42.6 to 49.2% top 3 range. Under the recorded device and technical circumstances, IntentSpider produces higher first rank and top 3 exact target word accuracy (measured) than tested versions of Google Gboard, Microsoft SwiftKey, Samsung Keyboard and OpenBoard. Opening the block and release process preserves all 500 event rankings while restoring 59 top 3 predictions from 94 undisplayed suggestion rows. Independent systems preserve this separation over 33000 paired rows and states. It separates personal ranking, display, learning and graph changes as well.

Keywords Human Computer Interaction· Graph Theory · Intent Graphs · Next Word Prediction· On device processing · Machine Learning

1 Introduction

In information systems, predictive text is an array of interations generated by a system that humans use to form, ignore or accpet decisions as predictions display consumes human attention. And personalization changes such as next word prediction affects adoption, cultivation or composition of their related decisions. [2,5,21,23]A complete prediction interface therefore, requires contsistant, personal data state, history/context/seed input construction, ranking, display control and the debugging ability.

IntentSpider WebNet addresses this end to end HCI question using a personal human intent graph.

One C++ main core uses humans, typing up to three next words, optional ability to accept, continued updates and HCI researchers who investigate the same graph state (publicly available

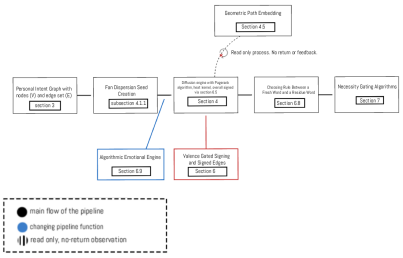


Fig. 1. An Overview of the IntentSpider processing architecture

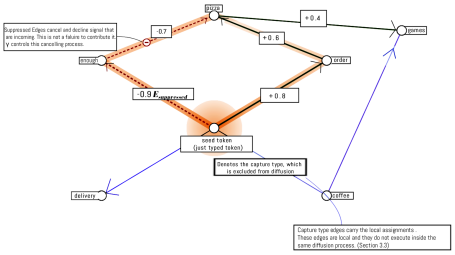


Fig. 2. Transmission Type (TF) and Capture Type roles of edges, including the diffusion path and direct association connections.

on intentSpider playground. Accassible via an API), saved states, event state record and 2D embedding without changing the ranking.

RQ1 How does this complete system process data during continuous value scoring and then enters to adaptation relative to the provided data references, commercial/existing keyboards and processing time? **RQ2** Which specific IntentSpider process process change the ranking or displaying most when one already saved state and target order sequence are re played with one controlled change? **RQ3** Do post ranking necessity gating and read only debugging satisfy their separation in the executable system without changing the ranking, and do the two separate display state systems preserve this condition point across independent systems?

Visibility invariance comes from this learning order and allows the necessity gate to be removed without changing the remaining prediction process.

The target words come from the released conversational data, not study messages produced by participants.

2 Related Work to IntentSpider

Current predictive keyboard systems use n gram or back off processes for personalization and local device or federated neural models [9,8,21,23,26], In addition to that, these type of keyboards, mostly used as writing assistants (such as automatic grammar checking) systems include generative artificial intelligence assistance [18,22]. IntentSpider, on the other hand, uses graph processes for creating the context, ranking data points, displaying results, recursive runtime calls and debugging separately instead of combining those executions into a one function call.

Displaying suggestions result in a storage and a processing infrastructure cost. And this ccan effect the selection, use and the writing behavior [21,2,5,23]. In studies related to human and system interaction, we should separate action confidence from action cost [10,6,7]. Ranking is completed before necessity gating. Replay based keyboard evaluation is already used in HCI research. [4], and direct word recommendation testing uses display as a change [24]. PageRank or local push algorithms [20,16,1] executes after the fan dispersion input seed is created. The 2D embedding answers specific debugging questions [19] and is used for debugging the fixed processing state, and not for identifying a medical or a psychological state. IntentSpider's novelty is a result of this complete webnet architecture. This includes but not limited to, a function typed personal intent graph, constructing context, ranking before necessity gating, continuous update actions and the 2D debugging.

3 A Novel, Personal Intent Graph with Function Typed Edges

Personal intent graph and function typed edges. The personal intent graph stores the prediction data state of one human, $G_t = (V_t, E_t)$ after event t, with the current words and

Table 1. Working configuration and controlled changes. Each replay reloads the recorded graph/configuration before EV0001. No coefficient is fitted during replay.

Process	Reference	Control
Function type	$TF_0 = .5,\ \theta_{TF} = .35$	reference setting not changed
Context	$k = 3$, fan dispersion seed	Most recent word used
Local push process	$\epsilon = 10^{-4}$, cap 200,000	LocalPush value is fixed
Growing radius	$\alpha_{effective} \geq .02$	diffusion radius is fixed
Residue choosing (prioritization)	$\rho = .5,\ c_{cal} = 1$	first and second word choosing is not used
Negative output	$\tau_{supp} = 7$ days, $\tau_{shock} = 30$ days	negative response removed
Grace period	$T_{grace} = 60$ s	grace period removed
Necessity gating	stored $\beta_{1,3}, \kappa_N$, after ranking	ranked words are returned without the display requirement

one to one connections When v is added after u, the connection value decreases as

$$w_{uv}(t + \Delta t) = w_{uv}(t)e^{-\Delta t/\tau_w} + \eta_w \mathbf{1}[u \to v]. \tag{1}$$

The raw coefficient stores the ordinary connection value. But transmission value is the decision executor for forwarding the diffusion.

For the pending prediction state, TF value uses only edges with diffusion mass. Above ϵ_{push}, the chosen word receives $o_{uv}(t) = 1$ and the other ranked words receive $o_{uv}(t) = 0$. The other remaining edges do not use an outcome value.

$$TF_{uv}(t + 1) = TF_{uv}(t) + \eta_{TF}\big(o_{uv}(t) - TF_{uv}(t)\big). \tag{2}$$

For every new connection, TF value begins at $TF_{uv} = 0.5$. The system uses $\theta_{TF} = 0.35$ as the margin cutoff of transmission. A value above this margin allows the connection to forward diffusion mass. If the value is less than or equal to the margin, it preserves the connection in the capture type, which does not forward diffusion mass. The graph learning process uses the complete ranked row for the TF update in "displayed" and "internal only" states. Because of this, the raw coefficient, TF, signed value, the decision to display is seperated from each other.

Fan dispersion seed and diffusion. The evaluated process starts the next-word prediction from the last $k = 3$ committed words. For each recent u, let $\pi_T(u, \cdot)$ be its normalized outgoing transmission edge distribution and, using Shannon entropy [25],

$$\phi(u) = \frac{H(\pi_T(u, \cdot))}{\log d_T(u)}, \qquad s_0 = \text{norm}\left(\sum_{i=1}^{k}(1 - \phi(u_{t-i+1}))e^{-(i-1)/\tau_{seed}} e_{u_{t-i+1}}\right). \tag{3}$$

Fan dispersion value is calculated only in moments a word has minimum of two external transmission connections. If that is 0 or 1, this does not add seed input mass for diffusion. The remaining prediction positions are allowed to use external connections from the latest or the most recent word. Before normalizing, this implementation checks the total value of seed input. If that value is zero, diffusion process execution is blocked, and the remaining positions are allowed to use direct external connections from the most recent word. Due to that reason, $\log 1$ does not create a forwarding value. Seed input mass increases when the dispersion value is lower and decreases when is greater. τ_{seed} controls the recent word decreasing logic.

For the transmission connections, $P_T = D_T^{-1} W_T$ The personalized distribution is

$$p = \alpha s_0 + (1 - \alpha)P_T^{\top} p. \tag{4}$$

The implementation uses local push [1]. After the recent context words are removed, the other remaining positive words are ranked by diffusion value. In moments where less than three ranked words are available, the remaining positions can use external connections from the most recent word. These associations do not forward diffusion.

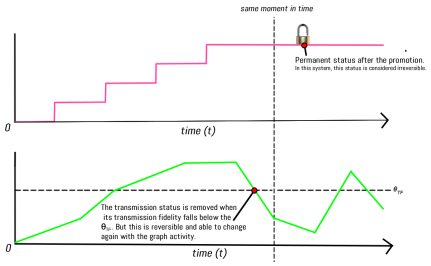


Fig. 3. Flexible TF state changes. These changes are two way. Reverse possible.

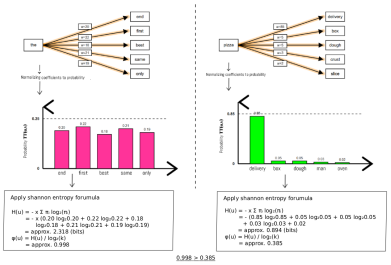


Fig. 4. High and low dispersion node connections and their Shannon entropy values.

The Typing Rate Difference Values and the Additional Runtime Processes. Typing interval and correction values produce the tone grade. The tone grade is a signed value from -1 to 1, calculated for one selection event using that human's rolling typing reference. This value is not a direct measurement of emotional meaning from the word. For the current average typing interval Δt_{avg}, rolling personal reference Δt_{ref}, and correction rate κ, the tone grade and signed edge update are

$$\begin{aligned} g_{tone}(c_t) &\equiv \mathrm{val}'(c_t) = \tanh\big(a(\Delta t_{avg} - \Delta t_{ref}) - b\kappa\big), \\ s_{uv}(t+1) &= \mathrm{clip}\Big(s_{uv}(t)e^{-\Delta t/\tau_v} + \eta_s g_{tone}(c_t), -1, 1\Big)\,. \end{aligned} \tag{5}$$

This signed value can build a negative history across earlier selection events. The raw coefficient is updated separately for the selected connection. This is also true in cases where same selection results in updates of both values.

Our system separately calculates the positive (unsigned) typing rate difference from the current typing rate compared to the ordinary typing rate of that human. This value is used for the hexarousal axis and does not have a specific direction such as positive or a negative. Hexarousal is not the arousal axis found in Professor Russell's research. When the typing difference (delta) is greater than the marginal value, the growing radius (A system process described earlier) increases the diffusion radius. In addition to that, IntentSpider applies compounding reinforcement and tests the sudden negative response. These measured typing patterns are also grouped as typing sub category states. The History Based Task Scheduling process applies the residue/support condition to the first and second ranked words. Then we introduce a grace period, the next engine event can adjust the stored temporary values, clear the related active state, and reset the typing rate streak. Table 1 denotes which processes are changed while the other processes continue.

Executable and Architecture The C++ version 17 core executes these processes directly [11]. New word sequences (strings) are stored locally. `predict()` function builds the fan dispersion seed input, executes local push algorithm, the ranked row is saved to the relevant storage and then executes the necessity gating algorithm. `accept()` processes that selected word through that same coefficient, signed value, and output updates used by ordinary typed input. Save function and load function preserves the current intent graph state. Because of that, those variables describe the same state changes used by the executable process.

4 Necessity Gating Algorithm and Suggesting Functions

The Necessity Value For the first ranked word w, the necessity value combines the confidence value from the diffusion output, the tone grade priority value and $x(w,t)$,

$$N(w,t) = \beta_1(1 - H_{norm}(p)) + \beta_2 \max(0, -g_{tone}(c_t)) - \beta_3 x(w,t). \tag{6}$$

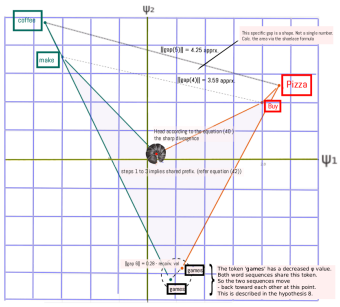


Fig. 5. Paths produced by many sequences of next word distributions in the intent graph.

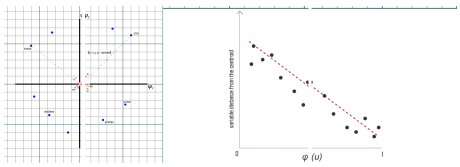


Fig. 6. DualNet two path embedding and the corresponding gap behavior

The system displays the ranked array of words for $N(w,t) > \kappa_N$. $x(w,t)$ is calculated from the recent context connections into the first ranked word. This process does not delete an edge and does not change its raw coefficient. The signed update processes the typing interval and correction data before the next word prediction. Due to that reason, the tone grade priority input is neutral in the necessity calculation that follows. Necessity gating executes after the graph ranking process. This specific order preserves the difference between the graph ranking process and the suggestion display process.

The runtime execution order also separates the suggestion display decision from the graph learning process. For one event, S_t represents the current graph state, x_t represents the current text input and y_t represents the selected word,

$$r_t = F(S_t, x_t), \qquad g_t = G(r_t, S_t), \qquad S_{t+1} = U(S_t, y_t, r_t). \tag{7}$$

The graph update processes r_t directly. The display result $g_t r_t$ does not enter this graph update.

The gate-open control sets

$$g_t = \mathbf{1}[r_t \neq \varnothing].$$

The 500 event controlled replay starts from the same S_0 and follows the same target word order. The same target word order produces the same text input at each event. The same graph state and the same text input produce the same ranked array of words. The same ranked array of words and selected target word then produce the same next graph state. This condition continues from one event to the next across the 500 events. Due to that reason, S_t and r_t remain the same across the complete controlled replay. The gate-open control changes the suggestion display while preserving the 500 event ranking sequence. This allows the display results to be compared without changing the ranking sequence.

Graph Learning Process. Section 6 also tests another condition. The learning process updates its state from the displayed result instead of the complete ranked array of words. A change in suggestion display then changes the later system state. This provides the comparison condition for the graph learning process.

The Display State Check. The controlled execution starts from the same state and follows the same ordered target words. The complete ranked arrays of words (rows) and the ranking state after each event also remain the same. These checks denote that suggestion display can change without changing the ranking sequence. Hence, the system comparison includes the state after each event and does not depend on the final accuracy result.

Geometric Paths and the 2D Embedding for Debugging. The ranking process reads the directed transmission graph. The 2D embedding uses a separate graph calculation. The transmission graph is converted into

$$A = (W_T + W_T^{\top})/2.$$

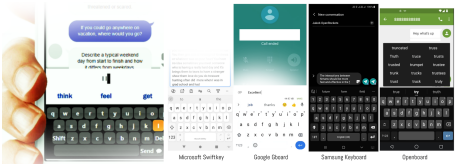


Fig. 7. Interfaces used during the evaluation and validation section. The first image is the IntentSpider Webnet web research preview followed with Microsoft SwiftKey, Google Gboard, Samsung Keyboard, and OpenBoard.

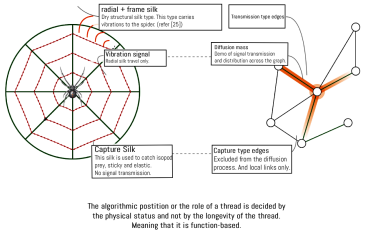


Fig. 8. The structure of the spiderweb and the correlated position of an edge or a thread or embeddings inside the main intent graph.

The largest connected component of this graph is used. D_A is its degree matrix and $L_A = D_A - A$. The two smallest nonconstant solutions of

$$L_A f = \lambda D_A f$$

provide the fixed 2D coordinate $z_v \in \mathbb{R}^2$ for each word node [3]. This 2D calculation does not change the directed transmission graph used by the ranking process.

For step i, m_i is the diffusion mass inside this component. Each $p_i(v)$ is divided by m_i before its 2D position is calculated,

$$m_i = \sum_{v \in L} p_i(v) > 0, \qquad \hat{p}_i(v) = \frac{p_i(v)}{m_i}.$$

The 2D position and the change from its previous position are

$$z(i) = \sum_{v} \hat{p}_i(v) z_v, \qquad \Delta z(i) = z(i) - z(i-1). \tag{8}$$

The ordered positions $z(1), \ldots, z(n)$ create a geometric path. Two word sequences with the same prior words, graph state and 2D embedding produce the same p_i and $z(i)$ for that shared prior words. After the two sequences use different next words, $z_A(i) - z_B(i)$ denotes the difference between their 2D positions. Later shared words can reduce this difference and move the two paths toward the same graph area again. This is the "reconvergence". Two diffusion outputs are allowed to share the same entropy and different 2D positions because their diffusion mass can be distributed across different word nodes.

The 2D embedding enables graph state for debugging. The reported checks cover the shared prior words, path separation and reconvergence, the same entropy at different 2D positions, and the execution of the 2D debugging process from the prediction. Task based human evaluation is denoted here as future work. It is separate from these system checks.

5 Final IntentSpider System and Its Public Results

Using the same C++ core for writing and research functions. The open C++ or standard library core [11] supports both roles. The writer path is raw keypress → tokenization → `predict()` → Top 3 → optional selection → continued update. A researcher can use live debugging values, state saving, replay, and the two dimensional embedding over the same predictor.The research functions are also included in the main engine code.

On Device Data Processing IntentSpider Web Research Preview executes Personal Mode and Global Person Mode through the same C++ core. Global Person Mode loads an already prepared graph so the system starts with existing graph data. The public branch stores this graph state through Cloudflare KV. This graph was prepared from public conversational material, including NVIDIA T4 GPU processing in Google Colab [13,14]. Personal Mode does not load this prepared graph. It builds the current human's graph from the base vocabulary and the entered interactions. The remote state process is optional and can load or save the graph state. After loading, the C++ engine executes ranking and continued graph updates inside the client device [14,15].

Vocabulary and Front End Support The word unit processing function converts the entered text into lowercase alphabetic words, numerical values and apostrophe words. New word strings are stored locally. The graph update and ranking processes process word identifiers instead of depending on one permanent word list [11]. Due to that reason, the engine is allowed to add new words and is not connected to one specific interface. The evaluation process uses English word units. Other native and web front ends can call the same `predict()` interface without changing the graph calculations. As future work, the same system can be evaluated with other languages such as Spanish and German.

Open Source and Result Files. The evaluation data is separate from the Playground training material. The Excel workbooks store the event rows for IntentSpider, Gboard, Samsung Keyboard, Microsoft SwiftKey and OpenBoard. The supporting files store the processing results, controlled condition results, repeated context results and the outputs from the independent implementations [12]. The released project data also preserves outputs from failed or abandoned executions. Source line and event IDs connect the reported measurements to their event rows. The controlled change configurations and scripts connect those event rows to the process changed in each condition. Due to that reason, the public source code, saved state, event rows, output files and reported tables connect the interface and C++ engine to the reported results and claims.

6 Experiments and Validation

6.1 Evaluation Data and Event Execution (Controlled)

The evaluation data contains a fixed short form English word sequence. This sequence remains in its original order for each controlled change [12]. Alphabetic words, numerical values and apostrophe words are treated as individual word units. Lines containing the project redaction marker or emoji characters are not used. The first word starts the input context. After that, IntentSpider predicts the next word. The evaluation compares the target word with the predicted words. IntentSpider then processes the entered target word and continues the graph update.

The 1,000 prediction events come from 65 parts of the source conversation. Those parts are contained inside nine continuous sections of the original source data. Each part contains 2 to 150 prediction events, with a median of 11. During this evaluation, one human enters the predetermined word sequence. The evaluation data specifies the target word for each event.

The human does not select suggestions during the controlled event executions. Because of that, suggestion display cannot change the later entered words. Keyboard evaluation with the same recorded word sequence is already used in HCI research [4].

The EV0001 to EV1000 sequence executes continuously without resetting the graph. EV0001 to EV0500 and EV0501 to EV1000 are two consecutive measurement sections. These sections are not a train/validation split. EV0501 to EV1000 continues after EV0001 to EV0500 without repeating a context, target or prediction record. EV1001 to EV1052 is stored separately. The starting state contains 40,096 observations. None of the first 500 targets was already used as an evaluation event inside that starting state.

The first and second 500 event sections contain 241 and 245 different target word types. Among them, 154 and 150 appear one time. Among the two word contexts, 94.3 and 93.5%

appear one time. Among the three word contexts, 98.4 and 99.0% appear one time. In addition to that, 84 of 89 and 85 of 91 repeated most recent word contexts are followed by more than one target word. The five repeated most recent word types in EV1001 to EV1052 have the same condition. Due to that reason, many exact longer contexts appear one time while a repeated most recent word can still be followed by different target words. This is the reason for testing the context depth change.

A keyboard data collection records Gboard, Samsung Keyboard, Microsoft SwiftKey and OpenBoard in their existing personalized states. The Samsung Keyboard collection uses a Galaxy A01 with Android 10 and Samsung Keyboard version 3.5.45.2. The Gboard, Microsoft SwiftKey and OpenBoard collections use a Pixel 4 XL with Android 10. Microsoft SwiftKey uses version 9.13.14.5. Samsung predictive text is enabled, while auto replace and spell correction are disabled. Gboard suggestions are enabled, autocorrection is disabled, and cloud personalization and synchronization remain enabled as a special test. Microsoft SwiftKey prediction is enabled, while autocorrect and emoji prediction are disabled. OpenBoard prediction is enabled. Human enters the same word sequence into each keyboard. The first three visible words are recorded before the target word is entered. A no suggestion row remains a valid scored event. Unreadable or "system error" output is not included in the related keyboard comparison. Each keyboard comparison includes each event row that has a valid result for IntentSpider and that keyboard. Hence, each result describes the complete displayed keyboard behavior under that recorded device and personalization state.

Ranking and display are separated. For complete row prior words $r_t^{(k)}$, display $g_t \in \{0,1\}$, and target y_t, define pre gate ranking $R_k = n^{-1} \sum_t \mathbf{1}[y_t \in r_t^{(k)}]$, display aware hit rate $D_k = n^{-1} \sum_t g_t \mathbf{1}[y_t \in r_t^{(k)}]$, coverage $C = n^{-1} \sum_t g_t$, and displayed correctness $Q_k = D_k / C$. If hidden rows are unavailable, the sharp bound is

$$D_k \le R_k \le \min\{1, D_k + 1 - C\}, \tag{9}$$

since every displayed hit is a ranking hit and hidden rows can add at most $1 - C$. Deployed keyboards report D_k. For display cost, let $H_k = nD_k$, $E = nC$, and $W_k = E - H_k$, then

$$U_k(\lambda) = H_k - \lambda E, \qquad U_k^W(\mu) = H_k - \mu W_k. \tag{10}$$

In here, the two equations count one prediction event as one opportunity. The first equation assigns a cost to each displayed suggestion row. The second equation assigns a cost to each displayed row that does not contain the correct word. Each equation reaches zero at the cost value where the correct suggestion benefit and the assigned display cost become equal. Therefore, the zero point of each equation produces its cost ratio for this comparison.

IntentSpider loads the previously recorded state and data configs into a new engine for each controlled execution. The experiment changes one system process and enters the 500 evaluation events again. This produces each process change one common starting graph state.

The reference condition and the gate open condition produce equal complete ranked arrays at each of the 500 events. The gate open condition displays 499 arrays that contain ranked words. The reference condition does not display 94 of those arrays. Therefore, the gate open ranking result produces R_k for the reference condition. The measured result in this paper uses the recorded 40,096 observation graph state. Another starting graph can run this comparison. The two executions must match the starting state, target order and graph update input.

Two additional self contained systems test the suggestion display and graph learning separation without IntentSpider code, graph state, parameters or graph processes. The first system predicts the next word from continuation counts after the most recent word in the fixed evaluation word collection. The second system selects alerts by adding one count to each possible alert result before calculating its probabilities. This system processes 32 previously generated alert sequences whose event frequencies change over time.

The paired gate conditions in each system process one target sequence and one complete ranked row update sequence. The validation compares the complete ranked arrays and the system state before and after each event (the implementation also stores hash values for these state comparisons). Another control connects suggestion display to the later system update. This control produces different later states. Because of that, the validation distinguishes a

Table 2. Top 3 counts/rates in adjacent windows, E/L=early/late, S/R=singleton/repeated target, B/O=branching/other latest word.

Stratum	EV0001–0500	EV0501–1000
Position	E 83/250 (33.20%), L 56/250 (22.4%)	E 31/250 (12.4%), L 84/250 (33.6%)
Recurrence	S 17/154 (11.04%), R 122/346 (35.26%)	S 40/150 (26.67%), R 75/350 (21.43%)
Latest word	B 65/338 (19.23%), O 74/162 (45.68%)	B 82/330 (24.85%), O 33/170 (19.41%)

display change that leaves graph learning unchanged from a display change that also changes graph learning.

The comparison systems begin with different earlier information. The IntentSpider time based execution and controlled executions use the recorded 40,096 observation graph state. The online most recent word count process starts from an empty state at EV0001. Kneser-Ney is an n-gram language model algorithm that redistributes probability across word histories [17]. The online Kneser-Ney process also starts from an empty state at EV0001. A second Kneser-Ney reference removes the scored event from its available evaluation word collection and processes the other continuations. This produces that reference more earlier word information than the online process. The existing keyboards retain their earlier personalization histories. The two additional self contained systems use unrelated starting states.

The complete evaluation contains 1,000 continuously updated events and a separate 52 event section. It also contains the online references, Kneser-Ney references, ten controlled process executions, 3,198 aligned keyboard rows, 33,000 paired events from the two additional systems, processing measurements and Playground execution.

Validation Results for the Research Questions. RQ1 uses the continuous event sections, reference systems, existing keyboard rows, measured processing time and Playground execution. RQ2 uses ten controlled process executions, the necessity gate comparison and the ranking bound. These measurements separate the complete ranking result, suggestion display result, correctness of displayed suggestions and the number of events with displayed suggestions.

The IntentSpider C++ engine is open source. In addition to that, the released materials include the saved state, Excel workbooks, controlled execution data configs, Windows and Bash scripts, Playground files, output files and records from failed or discarded runs. These materials connect the implemented system to the event rows and reported results [14,11,12].

7 Results From the Complete System and Controlled Changes

Keyboard Results and Behavior. IntentSpider produces 132 correct first rank events and 254 correct Top 3 events from EV0001 to EV1000. These results produce 13.20% first rank accuracy and 25.4% Top 3 accuracy. The system displays suggestions in 89.6% of the events. MRR@3 is .1825.

EV0001 to EV0500 produces 14.4% first rank accuracy and 27.80% Top 3 accuracy. EV0501 to EV1000 produces 12.00% first rank accuracy and 23.00% Top 3 accuracy. The graph continues updating through four consecutive 250 event sections. Top 3 accuracy in these four sections ranges from 12.4% to 33.6%.

The most recent word count reference predicts from continuation counts after the most recent word. It reaches 4.50% first rank accuracy and 6.50% Top 3 accuracy. The unigram backoff reference reaches 4.90% and 9.4%. The count reference with access to the evaluation continuation counts reaches 7.4% and 13.6%. The online Kneser-Ney reaches 6.6% and 11.70%. The Kneser-Ney reference that removes the scored event and processes the other available continuations reaches 8.50% and 15.00%.

These references do not begin from the recorded IntentSpider graph state. Hence, their results are reference measurements and are not process change comparisons from one starting graph state. Tables 2 3 provide the event section and keyboard results.

Table 3. IntentSpider and keyboard next word accuracy under the recorded device and personalization states. Each comparison uses the event rows available for IntentSpider and that keyboard. Δ produces the percentage point difference.

Product	N	First rank	Δ_1	Top 3	Δ_3
Gboard	394	17.77/7.61	+10.15	33.25/14.47	+18.78
Samsung Keyboard	934	14.03/0.64	+13.38	27.52/2.25	+25.27
Microsoft SwiftKey	955	14.14/2.20	+11.94	27.33/6.18	+21.15
OpenBoard	915	14.75/0.44	+14.32	28.09/5.68	+22.4

Table 4. Ten controlled executions from the recorded starting state. These are percentages.

Condition	D_1	D_3	C	Q_1	Q_3	R_1	R_3
Full reference	8.4	12.8	81.0	10.37	15.80	16.8	24.6
No loaded temporary state	8.2	12.8	81.0	10.12	15.80	[8.2,27.2]	[12.8,31.8]
Fixed keypress timing	8.8	13.2	81.8	10.76	16.14	[8.8,27.0]	[13.2,31.4]
No runtime tone grade update	8.6	13.2	80.2	10.72	16.46	[8.6,28.4]	[13.2,33.0]
Fixed diffusion radius	8.4	12.8	81.0	10.37	15.80	[8.4,27.4]	[12.8,31.8]
One word seed	31.4	42.6	93.4	33.62	45.61	[31.4,38.0]	[42.6,49.2]
No compounding reinforcement	8.6	12.8	80.2	10.72	15.96	[8.6,28.4]	[12.8,32.6]
No residue/support choosing	8.6	12.8	80.8	10.64	15.84	[8.6,27.8]	[12.8,32.0]
Gate open	16.8	24.6	99.8	16.83	24.65	16.8	24.6
No signed/suppression process	7.8	11.2	55.8	13.98	20.07	[7.8,52.0]	[11.2,55.4]

Changes to Ranking and Suggestion Display. Table 4 denotes the RQ2 results. The reference condition outputs $D_1/D_3/C = 8.4/12.8/81.0\%$. The gate open condition preserves the complete ranked arrays unchanged and outputs $R_1/R_3 = 16.8/24.6\%$. The one word context condition changes the recent word amount used by fan dispersion from three words to the most recent word. This condition produces $D_1/D_3/C = 31.4/42.6/93.4\%$ and $R_3 \in [42.6, 49.2]\%$. The lower limit of this R_3 range is 18.0 percentage points greater than the reference three word R_3 value of 24.6%. Due to that reason, suggestion display by itself cannot explain this increase. The ranking result also changes.

The $k = 1$ condition changes the fan dispersion input from three recent words to one recent word. TF, diffusion, the signed value process and the later system processes remain active.

The evaluation data also denotes why the most recent word does not identify one fixed next word. In EV0001 to EV0500, 84 of 89 repeated most recent word contexts, or 94.4%, are followed by more than one target word. In EV0501 to EV1000, this is also visible in 85 of 91 cases, or 93.4%. The five repeated most recent word types in EV1001 to EV1052 are also followed by more than one target word. And because of that, the $k = 1$ result cannot be explained by one repeated most recent word always producing one target word.

The 405 displayed arrays contain 42 correct first rank words and 64 correct Top 3 words. These values produce 10.37% first rank correctness and 15.80% Top 3 correctness inside the displayed group. The 94 arrays that are not displayed contain 42 correct first rank words and 59 correct Top 3 words. These values produce 44.68% first rank correctness and 62.77% Top 3 correctness inside the non displayed group.

The non displayed group therefore has 4.31 times the first rank correct-word rate and 3.97 times the Top 3 correct-word rate of the displayed group. The necessity gate does not display 50.0% of the available correct first rank results and 47.97% of the available correct Top 3 results. In comparison, it does not display 12.53% of the rows without a correct first rank word and 9.31% of the rows without a correct Top 3 word. The odds of finding the correct word in the non displayed group compared with the displayed group are 6.98 for first rank and 8.98 for Top 3. Under the recorded graph state and the direct target-word matching used in this experiment, the current necessity gate therefore removes correct ranked results at a greater rate than incorrect ranked results.

The runtime execution order processes the typing interval and correction data before the next prediction. Due to that reason, the next necessity calculation uses a neutral tone grade priority value. The largest difference from the reference is 0.4 percentage points for Top 3 and 0.8 percentage points for suggestion display.

In addition to that, suggestion display does not change the complete ranked arrays in the gate comparison. Hence, graph ranking and the current typing time input do not explain the 59 Top 3 correct words inside the 94 non displayed arrays. The necessity gate compares $\beta_1(1 - H_{norm}) - \beta_3 x(w,t)$ with κ_N in this event order. Future calibration can focus on the diffusion confidence term, the $x(w,t)$ term and the κ_N display margin instead of changing graph ranking or typing time processing.

The gate open condition adds 94 displayed arrays. These arrays contain 42 correct first rank words and another 17 correct words in rank 2 or rank 3. This produces 59 additional Top 3 correct words. The other 35 arrays do not contain the target word inside Top 3.

The rank value calculation produces a first rank target the value 1 and produces a rank 2 or rank 3 target the value $\omega \in [0,1]$. This produces

$$B(\omega) = 42 + 17\omega.$$

Dividing this value by the 94 additional displays produces

$$\lambda^* = \frac{B(\omega)}{94} \in [0.447, 0.628].$$

The second display cost calculation uses the additional displayed arrays without the related correct word. At first rank, 52 of the 94 arrays do not contain the target word in the first position. This produces $42/52 = 0.808$. At Top 3, 35 arrays do not contain the target word and this produces $59/35 = 1.686$. These are the two end values of this display cost calculation.

The condition without the signed value and negative output process reduces suggestion display from 81.0% to 55.8%. Its R_1 range is $[7.8, 52.0]\%$ and its R_3 range is $[11.2, 55.4]\%$. Due to that reason, the displayed suggestions by themselves do not produce the complete ranking change for this condition. The condition without the loaded temporary state changes one visible prediction row.

Results From the Two Systems. The most recent word count predictor produces equal ranked arrays and equal system states in all 1,000 paired events. The paired system states become different at event 2 in the display-linked update condition. The displayed group contains a Top 3 target in 18.82% of its events. The non-displayed group contains a Top 3 target in 8.61% of its events. Four of the five measured event groups have this direction. The first measured event group has a -0.97 percentage point difference.

The second system uses an alert selection process with smoothed earlier count values. It processes 32 previously generated alert event sequences with changing result distributions. The two display conditions produce equal outputs and equal system states across all 32,000 paired events. The display-linked update condition produces different later states in each of the 32 event sequences. The displayed result density is 76.32% and the non-displayed result density is 61.83%. This direction is opposite to the IntentSpider result.

The Processing Time for Keyboard Input and Next Word Prediction. The processing measurement uses Microsoft Edge on Windows 11 Pro with an Intel i7 1165G7 processor. A 90 second Playground typing workload contains 50 word boundary prediction events. The median processing time from keyboard input through the C++ prediction and suggestion display is 2.4 ms. The 95th percentile is 3.66 ms, the 99th percentile is 8.08 ms and the maximum is 11.8 ms. All 50 measured prediction events complete below 16.67 ms.

A second measurement uses a 30 minute human typing workload in a dedicated Microsoft Edge environment. The complete Edge process uses a mean 0.281% of the machine CPU. Its 95th percentile is 1.342% and its maximum is 2.458%. The saved graph occupies 2.209 MiB. The saved temporary IntentSpider state occupies 2.151 MiB and the WebAssembly program occupies 0.205 MiB.

8 Conclusion

IntentSpider is a next word prediction engine created according to a human's function typed personal intent graph.

Across EV0001 to EV1000, IntentSpider produces 13.20% first rank accuracy and 25.4% Top 3 accuracy. The one word context condition produces the greatest ranking related change and gives a calculated 42.6 to 49.2% Top 3 ranking range. The gate open condition again restores 59 internally hidden Top 3 correct words from 94 non displayed ranked arrays. The complete ranked arrays remain equal at all 500 events. The two independent systems preserve this display and graph learning condition across 33,000 paired rows and states. Their display linked controls produce different later system states.

Future work will study continued human and system adaptation with participants, repeat these measurements across other personal graph states, measure processing time and energy in native clients, change context depth during runtime, recalibrate the necessity margin values and evaluate the 2D debugging process through human tasks.

Due to that reason, IntentSpider provides one complete personal next word HCI system where personal graph state, context input, graph ranking, suggestion display, graph learning and debugging remain separate executable processes. The open C++ engine, Research Playground, saved state, event rows and output files provide the implementation and result data for these processes. [14,11,12].

References

1. Andersen, R., et al.: Local graph partitioning using pagerank vectors. In: Proc. FOCS (2006)
2. Arnold, K.C., et al.: Sentiment bias in predictive text recommendations results in biased writing. In: Proc. Graphics Interface (2018)
3. Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput. (2003)
4. Bi, X., et al.: Octopus: Evaluating touchscreen keyboard correction and recognition algorithms via remulation. In: Proc. CHI (2013)
5. Buschek, D., et al.: The impact of multiple parallel phrase suggestions on email input and composition behaviour. In: Proc. CHI (2021)
6. Chen, M.X., et al.: Gmail smart compose: Real-time assisted writing. In: Proc. KDD (2019)
7. El-Yaniv, R., Wiener, Y.: On the foundations of noise-free selective classification. J. Mach. Learn. Res. (2010)
8. Fowler, A., et al.: Effects of language modeling and its personalization on touchscreen typing performance. In: Proc. CHI (2015)
9. Goodman, J., et al.: Language modeling for soft keyboards. In: Proc. IUI (2002)
10. Horvitz, E.: Principles of mixed-initiative user interfaces. In: Proc. CHI (1999)
11. IntentSpider: Intentspider c++ engine source code. GitHub (2026)
12. IntentSpider: Intentspider evaluation data, workbooks, and supporting outputs. GitHub (2026)
13. IntentSpider Webnet: Conversational data used to initialize the global person research preview state. GitHub data (2026)
14. IntentSpider Webnet: Intentspider interactive research preview. Research Playground (2026)
15. IntentSpider Webnet: Intentspider research playground source and state service. GitHub (2026)
16. Jeh, G., Widom, J.: Scaling personalized web search. In: Proc. WWW (2003)
17. Kneser, R., Ney, H.: Improved backing-off for m-gram language modeling. In: Proc. ICASSP (1995)
18. Lee, M., et al.: A design space for intelligent and interactive writing assistants. In: Proc. CHI (2024)
19. Lim, B.Y., et al.: Why and why not explanations improve the intelligibility of context-aware intelligent systems. In: Proc. CHI (2009)
20. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Tech. Rep. 1999-66, Stanford Digital Library Technologies Project, Stanford University (1999)
21. Quinn, P., Zhai, S.: A cost-benefit study of text entry suggestion interaction. In: Proc. CHI (2016)
22. Reza, M., et al.: Co-writing with ai, on human terms: Aligning research with user demands across the writing process. Proc. ACM Hum.-Comput. Interact. (2025)
23. Roy, Q., et al.: Typing efficiency and suggestion accuracy influence the benefits and adoption of word suggestions. In: Proc. CHI (2021)
24. Schnabel, T., et al.: Recommendations as treatments: Debiasing learning and evaluation. In: Proc. ICML (2016)
25. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal **27**, 379–423, 623–656 (1948)
26. Xu, Z., et al.: Federated learning of gboard language models with differential privacy. In: Proc. ACL Industry Track (2023)