Cash Transaction Booking via Retrieval Augmented LLM

Xiaoli Zhang
Amazon
zhasabri@amazon.com

Daksha Yadav
Amazon
dakyadav@amazon.com

Boyang Tom Jin
Amazon
boyanjin@amazon.com

ABSTRACT

In large corporations, millions of cash transactions are booked via cash management software (CMS) per month. Most CMS systems adopt a key-word (search string) based matching logic for booking, which checks if the cash transaction description contains a specific search string and books the transaction to an appropriate general ledger account (GL-account) according to a booking rule. However, due to the free-text nature of transaction description and the diversity of cash transactions, CMS systems often fail due to data corruption (truncation, insertions, spelling errors), paraphrasing, and lack of reusable key word in the description, requiring significant manual intervention by accountants. Month over month, accountants manually handle CMS booking failures in spreadsheets. We present two machine learning models, a GL-account classification model and a search string extraction model, to alleviate this manual process. These two models, backed by retrieval augmented large language models, can automate booking for a substantial portion of the manual transactions. Our approach is robust to common data issues in transaction description. Unlike typical deep-learning models, our models are interpretable and explainable. For GL-account classification, our approach has an accuracy close to human experts. For search string extraction, compared to other methods such as fine-tuning transformers for extraction tasks, our approach produces reliable results closer to accountants.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**.

KEYWORDS

text classification, named entity recognition, deep learning

ACM Reference Format:
Xiaoli Zhang, Daksha Yadav, and Boyang Tom Jin. 2023. Cash Transaction Booking via Retrieval Augmented LLM. In *Proceedings of RoBust Fin (SIGKDD).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Similar to a personal bank account, corporations receive bank statements which record their cash activities such as invoice payments, disbursements, money transfers, and etc. Each cash transaction needs to be booked into the corporation's financial ledgers, typically using a cash management system (CMS). To book a cash transaction, the CMS checks if the transaction description (a free text field) contains any keywords ("search string") specified in the booking rules, and books the transaction to a specific general ledger account ("GL-account") accordingly. Based on the values of transactions in each GL-account, accountants prepare financial statements.

For large corporations with complex transactions, hundreds of thousands of booking rules are configured. Despite the extensive amount of booking rules, a considerable portion of transactions still fail all existing rules each month and need to be booked manually. Common root cause of failures including data corruption (truncation, insertions, spelling errors), paraphrasing, lack of standardized practices from various legal entities owned by the corporation, and lack of reusable keywords in the limited length of description. Each month, accountants download the unbooked cash transactions and manually find the appropriate search string and GL-account in spreadsheets. Synthetic examples of failed transactions and search strings identified by accountants are provided in Figure 1. For confidentiality, all identifiers and entity names presented in this paper are synthetically generated.

Date	Transaction Description	Search String (manually extracted)	GL-account (manually added)
2023/03/14	/ENTRY-14 MARTRF/REF 6009136E5A15/ORD/GREAT SOLUTION INC. /BNF/ /RFB/232606470 GRESQL/ROC/STARLING-0M3H2V5D0R4D3K9JKL/TRANSID/8M3Q2V5E10H4DK9JKL //20230113104029 3PAYMENT RECEIPT	6009136E5A15/ORD/GREAT SOLUTION	GL-account 1
2023/03/02	TDES:COMMISSION:CHARGES/FEESBREF:BN00500021CREF:1000cKCRREF:1000KCTL:1000cKC	TDES:COMMISSION:CHARGES/FEESBREF:BN00500021C	GL-account 44
2023/03/01	/ENTRY-01 MARMID/REF NE3402830BU40129/ORD/GOOD MORNING CAFE/BNF//INV/FR3940N5UJY 15.02.20232000129108	REF NE3402830BU40129/ORD	GL-account 123
2023/03/17	/ENTRY-17 MARTRF/REF 60220P19O9K13/ORD/ ABC BANK ASA VESTLAND /EU/5825 BERGEN /BNFINVOICE NUMBER EUIVVO11-278901	ABC BANK ASA VESTLAND	GL-account 1

Figure 1: Examples of transaction descriptions, search strings, and GL-accounts

Accountants look through the transaction description to identify appropriate search strings highlighted in orange. Due to lacking of reusable search strings, accountants often use entity names or search string with identifiers. Also, extraction pattern is often inconsistent among different accountants and practices might also change over time.

To alleviate the burden of this manual process, we propose two models for transactions that require manual bookings: one for GL-account classification and one for search string extraction. The model suggestions can then be reviewed, approved, or disapproved by accountants. Our models are trained using a private cash transaction dataset from a widely used CMS. Current models cover ~200 bank accounts and can alleviate ~80% of manual work. The proposed model is designed for semi-structured transaction description data with significant presence of identifiers and entity names and

SIGKDD, August 06–10, 2023, Long Beach, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

is robust to data issues such as truncation, insertions, spelling errors, paraphrasing, and inconsistent extraction pattern. It has an accuracy of 99.7% relative to a human expert[1].

2 RELATED WORK

Identifying GL-accounts using transaction descriptions can be framed as a text classification problem. One prevalent solution to text classification is to fine-tune pre-trained LLMs ([6], [12], [15]). However, the performance of fine-tuned models drop significantly when annotated data is scarce, especially for minority classes ([5], [17]). The problem of extracting search strings can be framed as a few-shot named entity recognition (NER). Most search strings extracted by humans are entity names that represent buyers or sellers, identifiers that represent systems processing the transaction, or key words that describe the nature of the transaction (e.g. "commission", "expense"). Few-shot learning poses a challenge when limited examples are available for most classes. One popular solution proposed to resolve this is a prototype network ([18], [9]). Prototype methods learn the embedding of each entity type. It uses metric learning during training and nearest neighbor criterion to assign entity types during inference. However, since entity labels (e.g. identifier type) are not available in our dataset, this solution is not directly applicable.

With the rise of generative LLMs, recent work has shown that many NLP tasks can be solved by in-context learning without tuning model weights. [3] proposed an in-context learning approach using LLMs to solve tasks following a few demonstrations presented in the prompt. [11] proposed a KNN-augmented in-context example (KATE) selection strategy to select examples for the prompt. Similarly, [8] proposed the REALM model architecture that used a customized retriever tailored to the task of open-domain question and answering. In the NER domain, [7] explored few-shot NER via in-context learning using GPT-2 ([16]).

Our work follows the design of retrieval augmented LLMs ([8]) and combines metric learning adopted in prototypical networks with in-context learning via generative LLMs. Unlike [11] and [7], who used embeddings generated by pre-trained LLMs to retrieve demonstrative examples, we train a customized encoder via metric learning. Unlike [8], our retriever is optimized for the task of cash transaction booking and can handle semi-structured text data with significant presence of acronyms and identifiers. Contrary to prototypical networks, our approach does not require any entity type labels.

2.1 Dataseet

We use transactions from a commercial CMS to train and test our model. Transactions from Nov and Dec 2022 are used for model training and transactions from Jan 2023 are used for evaluation. In addition to description, each transaction also has attributes such as the amount, currency, date, and transaction type. We limited the scope of the model to the top ~200 bank accounts by volume that accounted for ~80% of the transactions manually booked in Jan 2023. Each of these transactions were booked to one of the ~2000 GL-accounts using one of the ~370k search strings. The distribution of GL-account and search string is highly imbalanced. Over 50% of GL accounts have less than 50 transactions per month and 70%+ of search strings can only match one transaction in the training and testing dataset.

To test the performance of our proposed model architecture on close-source SOTA models (e.g. OpenAI models), we also built a synthetic bank transaction description dataset (Appendix E), which contains 5,000+ transactions. Like real-world transactions, these transaction descriptions are semi-structured and dominated by identifiers (e.g. various transaction IDs coming from ERP systems or from the bank side). Synthetic data follows a key-value format (e.g. ORG="AAA Incorporated", BNF="BBB Solution"); we deliberately omitted the key or value in random cases to simulate inconsistencies between banks. We share experiment results using the private dataset as well as the synthetic dataset.

[1]Human experts cannot achieve 100% accuracy in GL-account classification task just by reviewing transaction description. In some cases, information presented in transaction description might be insufficient. Experts will contact relevant parties, e.g. source team, bank, to resolve ambiguities

3 MODEL DESIGN

3.1 Inference Architecture Overview

Intuitively, GL-account and search string can be determined by looking at how similar transactions are booked in the history. Based on this intuition, we built two models, one for GL-account classification and one for search string extraction. Both models follow a two-step process: (1) retrieve similar historical transactions and (2) make predictions by following historical practices. For any given new transaction, K-Nearest Neighbour (KNN) retrieves the top K most similar historical transactions, the associated search string, and the GL-accounts. To classify GL-account, we simply take the majority vote over the retrieved historical transactions. To extract search string, we compile a prompt and use a Generative Large Language Model (LLM) to perform search string extraction via in-context learning. We post-process the response from the Generative LLM to remove extra contents, address limitations, and confirm the validity of the extracted search string. To define transaction similarity, we train a multi-modal transaction encoder via metric learning. The transaction encoder generates transaction embeddings, which are used to measure the distance between transactions for both models. Figure 2 illustrate the intuition of the model design. Figure 3 provides the overview of inference architecture.

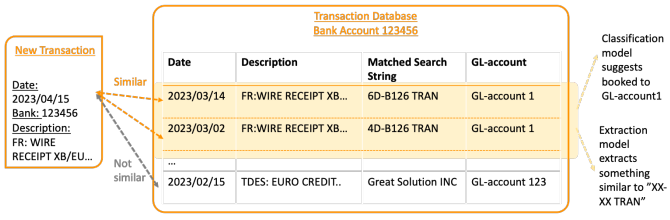

Figure 2: Intuition of how our models work

The new transaction on 2023/04/15 is similar to two transactions in the historical transaction database, but is dissimilar to the rest. Since these two historical transactions were booked to GL-account 1, the model suggests booking to account 1 for the new transaction.

As illustrated in Figure 3, both models have two phases: the encoding phase and prediction phase. During the encoding phase, historical cash transactions are encoded and saved to a database along with the ground truth GL-account and search strings used for booking. In the inference phase, we first call the encoder to

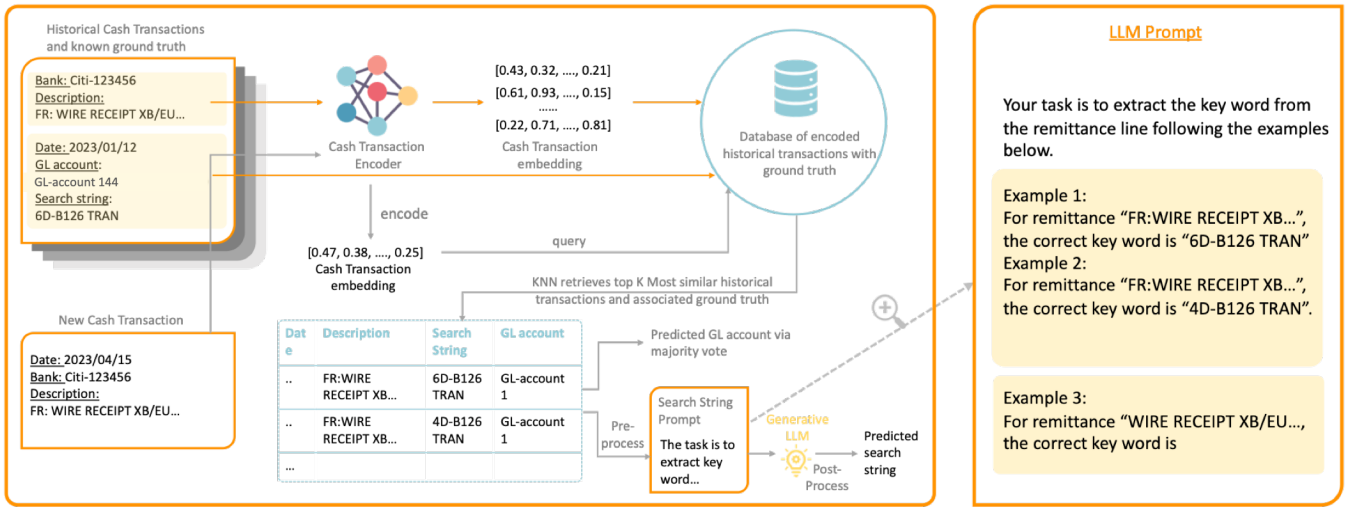

Figure 3: Overview of model inference architecture

In the encoding phase, historical transactions are encoded and saved to a database along with the ground truth GL-accounts and search strings used for booking. In the inference phase, we first call a customized encoder to obtain the embedding of the new transaction, then use KNN to retrieve top K most similar transactions from the database. To classify GL-account, we simply take the majority vote over historical GL-accounts. To predict search string, we compile a prompt, using historical transactions and search strings as demonstrations, and ask the Generative LLM to extract the search string for the new transaction. The prompt follows next-work-prediction format, compatible with most open-source LLM.

obtain the embedding of the new transaction, then use KNN to retrieve top K most similar transactions from the database based on the L2 distance between the embedding of new transaction and those of historical transactions. To classify GL-account, we simply take the majority vote over GL-accounts of the retrieved historical transactions. To extract search string, we compile a prompt, using historical transactions and search strings as demonstrations, and ask the Generative LLM to extract the search string for the new transaction. Pre and post processing is added to address limitations and validate the search string extracted by LLM. The following sections explain the major components in the model inference architecture.

3.2 Transaction Encoder

As presented in Figure 4, transaction encoder is a multi-modal encoding module with late fusion. It takes text features, categorical features, and numerical features from a transaction and outputs an embedding. Text features are concatenated with a fixed prompt and processed by pre-trained and fine-tuned LLM; categorical features are mapped to embeddings (learned during training) and numerical features are pre-standardized. The linear projection of the sentence embedding generated by LLM is concatenated with categorical features and numerical features, then processed by a single MLP layer. The output of the MLP layer is the final transaction embedding. We use metric learning with triplet loss to train the encoder. Multiple negative sampling logic is applied to facilitate learning.

3.2.1 Loss Function for Transaction Encoder. To train the transaction encoder, we use metric learning with triplet loss. Metric learning, instead of a softmax classification head, is used because of the following concerns:

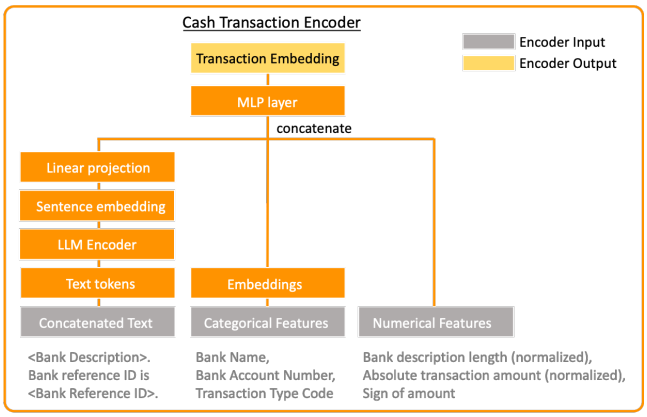

Figure 4: Transaction encoder architecture

Transaction encoder is a multi-modal encoding module with late fusion. It takes text, categorical and numerical features and outputs embedding. We use metric learning with triplet loss to train the encoder.

(1) Reusability of the encoder module. The search string extractor also relies on transaction embeddings to perform KNN retrieval and prompt compilation.
(2) High number of GL-account class labels (~2000) and the extreme imbalanced nature of the data distribution across different categories. Metric learning tends to perform better on rare classes with limited examples. Appendix A.1 provides details of our experiment using classification approach.
(3) Training time. On image classification tasks, metric learning tends to take less training time compared to classification ([10]). We observed similar pattern in our experiments (Appendix A.1).

(4) Scalability concern over re-training for new GL-accounts, new bank accounts, and changes in accounting practices. New GL-accounts or bank accounts could be added due to regional expansion, new business or merger and acquisition, and accounting practices could change overtime due to regulation changes. With the classification approach, the classifier would need to be re-trained. With the encoder + KNN approach, encoder re-training is not mandatory. Model could perform zero-shot inference as long as similar transactions are encoded in the database. Appendix B provides more details on zero-shot performance of the encoder.

3.2.2 Sampling Logic for Triplet Generation. Triplets are sampled according to the GL-accounts (class label). Transactions booked to the same GL-accounts are positive pairs, while transactions booked to different GL-accounts are negative pairs. To facilitate efficient learning, we adopt 2 rounds of negative mining strategy, one before we start training the model, one after we train the model with 1 epoch of data.

The first round is simple negative mining based on shared categorical features. Hard-negatives are transactions that share similar attributes, but eventually booked to different GL-account. We up-sample negative transactions pairs that come from the same bank account and share the same transaction type. We do not follow the distance based negative-mining techniques mainly due to concerns overs the accuracy of distance metrics. Since transaction description contains large amounts of IDs and entity names (90%+ in certain cases), distance metrics can be highly inaccurate.

The second round of negative mining is based on model errors evaluated on training data. After 1-epoch of encoder training, we generate hard negatives based on model prediction errors. For any given anchor transaction, the encoder encodes the transaction and KNN retrieves the top K similar transactions. Within the retrieved transactions, if the transaction is booked to a GL-account other than ground truth anchor GL-account, we consider the historical transaction as a hard negative. A L2 distance filter is also applied.

3.2.3 Large Language Model. We use a pre-trained BERT ([6]) as the base language model for transaction encoder. We further fine-tune BERT model on the masked-language-model (MLM) task on transaction description. All parameters were unfrozen during MLM fine-tuning and encoder training. However, ablation analysis (Appendix A.2) shows fine-tuning BERT on MLM task have limited impacts on the final model performance.

3.3 Search String Extractor

As shown in Figure 5, search string extractor relies on the Generative LLM to perform in-context learning. Pre-processing and post processing are added to address limitation of the model response. To validate the search string extracted by the model, we perform validation by back-testing the new search string using historical transactions.

3.3.1 Generative Large Language Model. We rely on a pre-trained generative LLM to perform in-context learning for search string extraction. To select an appropriate generative LLM, we tested the following models using the prompts listed in Appendix D: FLAN-UL2([20]), FLAN-T5-XL([4]), FLAN-T5-XXL([4]), BLOOM-7B([2]), BLOOM-176B([2]), GPT-J-6B([21]), and Alexa-20B([19]). Among all of the models tested, only BLOOM-176B and Alexa-20B were able to generate correct predictions in all samples; FLAN-T5-XXL and FLAN-UL2 (X-denoising head) made 1 mistake while none of the other models generated any reasonable results. Since BLOOM-176B and Alexa-20B are the two biggest model among all models tested, we conclude that model scale is a key factor that drives the quality of in-context extraction results. We used Alexa-20B for all major experiments. [2]

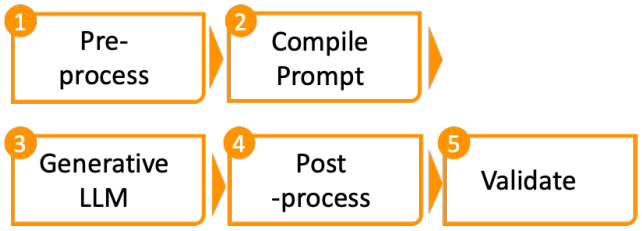

Figure 5: Architecture of search string extractor

Extractor mainly leverages in-context learning capability of generative LLM. Pre and post processing are added to address limitation of the model response. We also back-test the extracted search string using historical transactions.

3.3.2 Limitations of the Search String Extractor . We noticed four types of errors with the search strings generated by Alexa-20B. Pre-processing and post-processing modules are added to address those limitations. We verified that those issues were not specific to Alexa-20B via sample testing using BLOOM-176B.

(1) **Spacing**. Non-standard spacing, such as tab or double-space are sometimes replaced by a singe space in the model output. For example, the correct search string is "AAA<tab>SOLUTION", while model outputs "AAA<space>SOLUTION". Sample testing shows BLOOM-176B also shares the same issue. To fix this issue, we use a post-processing rule based on the location mapping between text with spaces and text without spaces to recover the correct search string.
(2) **Casing**. Most bank descriptions in our dataset are sent in all upper-case letters. However, in cases where lower-case is used, Alexa-20B tends to auto-capitalize the extracted search string regardless of the original casing. For example, the correct search string is "AAA Solution Inc.", while the model outputs "AAA SOLUTION INC.". Sample testing shows BLOOM-176B also shares the same issue. We use a post-processing rule to fix the casing.
(3) **Spelling correction**. Bank description could contain spelling errors or truncated words. Alexa-20B tends to auto-correct the spelling issues. For example, the correct search string is "BBB HEALTHCAR. INC", while the model outputs "BBB HEALTHCARE. INC". We use post-processing rule to fix the issue by adopting the longest matching string between the search string extracted by the model and the original transaction description.
(4) **Long-range dependency**. It is observed that Alexa-20B tends to generate unreasonable responses when the transaction description is long and the correct search string is located at the

[2]We want to balance the trade-off between model size and capability, so we chose the smallest model that passed our prompt tests. To test the stability of model output, each candidate model is invoked multiple times under varying temperatures.

front or middle of the description. Similar issues are observed on BLOOM-176B. We believe the issue is related to long-range dependency. Intuitively, attention in transformers are rather localized and models generally see limited associations between words that are very far away from each other in pre-training. To address this issue, we use pre-processing rules to truncate long description where we tend to see high error rate. Only 2 rules are configured as most failures follow 2 formats. However, we recognize the scalability risk of this rule-based fix and will explore other solutions. Appendix C provides examples and more experiment results related to this finding.

3.3.3 Validation of extracted search string. Search string extracted by the model should only map to a single GL-account. We validate the search string by back-testing using historical transactions. Specifically, we locate all historical transactions stored in the private transaction database that contain the proposed search string and confirm if those historical transactions were booked to the same GL-account. If not, the proposed search string would lead to booking errors, thus we treat it as failing validation.[3]

4 EVALUATION RESULTS - PRIVATE DATASET

4.1 Evaluation of Transaction Encoder - Private Dataset

As it is hard to evaluate the quality of embedding directly, we rely on the performance of downstream tasks to evaluate the encoder. For GL-account classification, we use classification accuracy as the metric. We also benchmarked the performance of our customized encoder with out-of-the box encoders using pre-trained LLMs, such as pre-trained BERT and pre-trained FLAN-T5. For confidentiality, we disclose the accuracy relative to human expert in Table 1. Evaluation results for search string extractor are covered in 4.2.

Table 1: Embedding Quality - Measured via accuracy of GL-account classification task (Our private dataset)

Model	Accuracy
Human	100%
Ours	99.7%
Pre-trained BERT	6.8%
Pre-trained fine-tuned BERT*	7.0%
FLAN-T5-small-mean-enc	8.3%
FLAN-T5-small-enc-dec	6.7%
FLAN-T5-xl-mean-enc	7.1%
FLAN-T5-xl-enc-dec	7.3%

* Pre-trained BERT fine-tuned on MLM task using transaction descriptions.

Our trained encoder + KNN achieved an accuracy of close to human performance on GL-account classification task, while all other out-of-the box LLMs are 10 times less accurate compared to our model. For a fair comparison, all features used by our transaction encoder are concatenated in the format of "<Feature name 1> is <Value 1>. <Feature name 2> is <Value 2>..." and used for generating text embeddings. We set K to 3 for KNN. For FLAN-T5, we used two types of embeddings: average embedding value from the encoder ("mean-enc") and embedding value of the first output of decoder ("enc-dec"), following the work of [13]. As an out-of-the-box encoder does not know the key words and patterns that determine the GL-account, none of them achieve desirable accuracy.

For our transaction encoder, we also conducted a manual sample review of the model error. Based on the sample review and discussions with accounting experts, we determined that most errors are due to insufficient information in the transaction description. In these cases, human-to-human communication is required to make appropriate booking decision[4]. The remaining errors are mainly cases where very few training data of similar format (less than 5, often 0 or 1) are available.

4.2 Evaluation of Search String Extractor - Private Dataset

Search string extraction is subjective in nature. Even though there is some tribal knowledge for selecting representative search strings, different accountants may follow different practices and practices also change over time. For the example description "REFERENCE <ID1> <ID2>/BNF/<seller name>/...", while an accounting expert might recommend extracting <seller name>, some accountants might still choose <ID1> or even <ID2>/BNF/<seller name>. To accommodate the subjective nature of search string extraction, we use four metrics to evaluate the extractor:

(1) **Ratio of usable search string**. A usable search string is a search string that exists in the transaction description and validated by historical transactions (as described in section 3.3.3[5]). Ratio of usable search string = count of usable search strings/total number of transaction descriptions.
(2) **Ratio of re-usable search string**. A re-usable search string is a search string that exists in the transaction description, validated by historical transactions and can match more than one historical transaction. Ratio of re-usable search string = count of re-usable search strings/total number of transaction descriptions.
(3) **Ratio of exact match**. Count of (search string extracted by model = search string extracted by accountants)/total number of transaction descriptions.
(4) **Jaccard distance**. We use Jaccard distance to measure similarity between search string extracted by the model and that extracted by accountants. [6]

[3] If the search string suggested by the generative model failed the validation rule, model would, by default, output the entire description as the search string. This approach is not desirable although still valid.

[4] Due to insufficient information in the bank description, human experts cannot achieve 100% accuracy just by reviewing bank description

[5] If a search string fails the validation rule, we do not count it towards usable ratio. In theory, we can always use the entire transaction description as the search string. In practice, this is not recommended by accounting experts and should be treated as a last resort. For example, if GL-account classifier provides the GL-account, but search string extractor fails to provide valid search string, we default the search string to the entire transaction description.

[6] For example, description is "...AAA SOLUTION INC. USA...", model extracted "AAA SOLUTION INC", accountants extracted "AAA SOLUTION INC. USA". Both search strings are seller names, yet differ slightly in length.

The first two metrics measure if the extracted search string can be used or re-used; the last two metrics compare the similarity between model-extracted search string and human-extracted search string.

We also train an extractor using pre-trained BERT as benchmark. BERT-extractor uses token-level label and is trained with cross-entropy loss. At a token-level, 0 means not extracted and 1 means extracted. Since the ground truth search string requires a letter-level label and pre-trained BERT operates at token-level, we set label to 1 if we run into label conflict. For example, description = "REFER Solution Inc", tokens = ["REFER", "Solution", "Inc"], ground truth search string = "ER Solution Inc". We set label for "REFER" to 1 even though "REF" was not in the ground truth.

Table 2: Evaluation results of search string extractor (accuracy relative to our model)

	Metrics			
	Usable	Re-usable	Exact-match	Avg Jaccard distance
Ours	**100.0%**	100.0%	**100.0%**	**1.00**
BERT-0.01	26.8%	7.3%	5.6%	1.81
BERT-0.03	38.1%	13.3%	4.6%	1.87
BERT-0.05	38.1%	33.5%	6.0%	2.00
BERT-0.07	44.4%	88.2%	5.0%	2.13
BERT-0.09	51.3%	**109.9%**	3.1%	2.27

Table 2 presents the evaluation results. For confidentiality, we disclose the accuracy relative to our model. Since BERT extractor outputs probabilities, we evaluate the BERT extractor at different probability thresholds. As we increase the probability threshold, we observe an increase in re-usable and usable ratio, yet decrease in the similarity between results extracted by BERT and by accountants. Our approach significantly outperforms BERT extractor at almost all probability thresholds except the re-usable ratio from BERT-0.09. However, for BERT-0.09, many extracted search strings failed validation process, thus not usable. Compared to our approach, BERT was not able to follow the patterns of how accountants select search string. Sample review shows BERT tends to fail when extraction pattern is inconsistent. On the other hand, when generative LLM is presented with inconsistent practices in the prompt, it tends to follow one of them. Appendix F provides sample extraction results from BERT and Alexa-20B.

5 EVALUATION RESULTS - SYNTHETIC DATASET

Due to confidentiality concerns, we did not perform experiments using closed-source SOTA models on the private dataset. As an alternative, we benchmark with OpenAI SOTA models using synthetic bank transaction data (Appendix E). We also evaluate the robustness of our approach by explicitly ingesting common data formatting issues, inconsistent human extraction labels (4.2) and long-range dependency issues (3.3.2) into synthetic data.

5.1 Evaluation of Transaction Encoder - Synthetic Dataset

We evaluate embeddings generated by OpenAI text-embedding-ada-002 on the task of GL-account classification.[7] Similar to other pre-trained LLM's shown in 4.1, text-embedding-ada-002 does not perform well. Embeddings generated by text-embedding-ada-002 achieved an accuracy of 0.5%. Distance between embeddings generated by text-embedding-ada-002 is not reflective of how those transactions are booked.

5.2 Evaluation of Search String Extractor - Synthetic Dataset

We evaluate the proposed search string extractor using open-source and close-source generative LLM's. As a baseline, we follow the same prompt format as illustrated in Figure 3 and ensure Example 1 and Example 2 follow the same description and extraction pattern. To test the robustness of our approach, we explicitly ingest noise to our prompts. Specifically, we quantify the impacts of the following data issues:

- **Text corruption**. We randomly ingested formatting issues (insertion of space, missing delimiter) into 30% of the descriptions.
- **Inconsistent demonstrations**. We conducted two experiments. In the first experiment ("irrelevant example"), we keep the first demonstration and replace the second demonstration with an irrelevant example[8]. Experiment one mimics the scenario where KNN retrieves an irrelevant example. In the second experiment ("inconsistent search string"), we keep transaction descriptions in both demonstrations, but modify the search string of the second demonstration. Experiment two simulates the inconsistency in the search strings extracted by human (4.2).
- **Long-range dependency** (discussed in 3.3.2). We triple the length of a typical transaction description by appending random address, bank name and ~20 random identifiers of various formats. We deliberately keep the correct search string at the beginning of the description.

We use OpenAI text-davinci-003 and Alexa-20B in our experiments. We only adopt the exact match ratio, which measures if the search string extracted by the model is the same as search string defined by the rule. We do not apply pre-processing or post-processing described in 3.3.2. Table 3 shows the experiments results.

As shown in Table 3, both Alexa-20B and Danvinci-003 are generally robust to text corruptions and inconsistent demonstrations. Danvinci-003 is also robust to the long-range dependency issue. For Alexa-20B, we would need to rely on the pre-processing described in 3.3.2 to address the limitation on long description. For "irrelevant example" test, we further tested model performance with a single correct demonstration. Using Alexa-20B, 89% of errors are corrected if a single correct demonstration is used. For Davinci-003, all errors are corrected. This implies having less yet highly relevant examples are better than having irrelevant examples for in-context extraction task.

[7] According to OpenAI Blog post([14]), text-embedding-ada-002 performs the best on text classification and text search tasks among all OpenAI text embedding models.
[8] We replace example 2 with a description and search string from a different GL-account. The format of the description in Example 2 is quite different from example 1 and the query.

Table 3: Search String Extraction Results Using Synthetic Transaction Dataset - Exact Match Ratio

	Baseline	Text corruption	Irrelevant example	Inconsistent search string	Long description
Alexa-20B	99.8%	99.8%	97.0%	98.1%	33.2%
Davinci-003	100.0%	100.0%	98.0%	99.3%	100.0%

6 CONCLUSION

We present two models, GL-account classifier and search string extractor, following a retrieval augmented large language model architecture to alleviate manual booking currently performed in spreadsheets. For GL-account classification, our approach has an accuracy close to human experts despite the extreme imbalanced distribution of the class labels. For search string extraction, compared to other popular methods such as fine-tuning pre-trained transformers for extraction tasks, our training-free approach produces results that are more reliable, and closer to accountants. Both models are robust to data issues such as insertion, missing delimiter, paraphrasing and inconsistent extraction labels. Unlike typical deep-learning models, our model results are highly interpretable and follow the same method as how human makes booking decisions. We also present the similar transactions extracted by KNN to the accountants, which facilitates the human review process.

REFERENCES

[1] AI21. 2023. *AI21.* https://docs.ai21.com/docs/jurassic-2-models

[2] BigScience Workshop. 2022. BLOOM (Revision 4ab0472). https://doi.org/10.57967/hf/0003

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[4] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling Instruction-Finetuned Language Models. https://doi.org/10.48550/ARXIV.2210.11416

[5] Danilo Croce, Giuseppe Castellucci, and Roberto Basili. 2020. GAN-BERT: Generative adversarial learning for robust text classification with a bunch of labeled examples. In *ACL 2020.* https://www.amazon.science/publications/gan-bert-generative-adversarial-learning-for-robust-text-classification-with-a-bunch-of-labeled-examples

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[7] Elena V. Epure and Romain Hennequin. 2022. Probing Pre-trained Auto-regressive Language Models for Named Entity Typing and Recognition. arXiv:2108.11857 [cs.CL]

[8] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: Retrieval-Augmented Language Model Pre-Training. arXiv:2002.08909 [cs.CL]

[9] Jiaxin Huang, Chunyuan Li, Krishan Subudhi, Damien Jose, Shobana Balakrishnan, Weizhu Chen, Baolin Peng, Jianfeng Gao, and Jiawei Han. 2020. Few-Shot Named Entity Recognition: A Comprehensive Study. arXiv:2012.14978 [cs.CL]

[10] Jongpil Lee, Nicholas J. Bryan, Justin Salamon, Zeyu Jin, and Juhan Nam. 2020. Metric Learning vs Classification for Disentangled Music Representation Learning. arXiv:2008.03729 [cs.SD]

[11] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What Makes Good In-Context Examples for GPT-3? arXiv:2101.06804 [cs.CL]

[12] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]

[13] Jianmo Ni, Gustavo Hernández Ábrego, Noah Constant, Ji Ma, Keith B. Hall, Daniel Cer, and Yinfei Yang. 2021. Sentence-T5: Scalable Sentence Encoders from Pre-trained Text-to-Text Models. arXiv:2108.08877 [cs.CL]

[14] OpenAI. 2023. *New and Improved Embedding Model.* https://openai.com/blog/new-and-improved-embedding-model/

[15] XiPeng Qiu, TianXiang Sun, YiGe Xu, YunFan Shao, Ning Dai, and XuanJing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences* 63, 10 (sep 2020), 1872–1897. https://doi.org/10.1007/s11431-020-1647-3

[16] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[17] Yiwen Shi, Taha Valizadehaslani, Jing Wang, Ping Ren, Yi Zhang, Meng Hu, Liang Zhao, and Hualou Liang. 2022. Improving Imbalanced Learning by Pre-finetuning with Data Augmentation.

[18] Jake Snell, Kevin Swersky, and Richard S. Zemel. 2017. Prototypical Networks for Few-shot Learning. arXiv:1703.05175 [cs.LG]

[19] Saleh Soltan, Shankar Ananthakrishnan, Jack FitzGerald, Rahul Gupta, Wael Hamza, Haidar Khan, Charith Peris, Stephen Rawls, Andy Rosenbaum, Anna Rumshisky, Chandana Satya Prakash, Mukund Sridhar, Fabian Triefenbach, Apurv Verma, Gokhan Tur, and Prem Natarajan. 2022. AlexaTM 20B: Few-Shot Learning Using a Large-Scale Multilingual Seq2Seq Model. arXiv:2208.01448 [cs.CL]

[20] Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Siamak Shakeri, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Denny Zhou, Neil Houlsby, and Donald Metzler. 2023. UL2: Unifying Language Learning Paradigms. arXiv:2205.05131 [cs.CL]

[21] Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

A ABLATION ANALYSIS

A.1 Classification Approach vs Metric Learning Approach

We compared the performance of classification approach using cross-entropy loss with metric learning using triplet loss for GL-account classification task. We conducted an experiment on 20 bank accounts where a classification head was added to the encoder and the model was trained with cross-entropy loss. When trained until convergence, we observe less than 50% accuracy on GL-account classification task on bank accounts with less than 30 training data points despite up-sampling. With metric learning (trained on the same 20 bank accounts), similar issues were only observed on bank accounts with fewer than 5 training data points. Also, we find it faster to train via metric learning. Given the same amount of training time and the max batch size allowed by the same hardware, models trained via classification approach did not reach convergence with the training loss continuing to trend down. Evaluation results confirmed the classifier at that checkpoint had less than 30% accuracy.

A.2 Fine-tune LLM using masked language modelling

We conducted a benchmarking experiment on the GL-account classification tasks using a pre-trained BERT and pre-trained BERT fine-tuned on the masked-language-model task on transaction descriptions. We fine-tuned BERT for 2 epochs. We confirmed the

training loss is reasonable and the prediction for the masked token is reasonably correct if the masked token is not part of an identifier. However, as shown in Table 1, we observe limited lift in the accuracy of GL-account classification task between BERT and fine-tuned BERT. We believe this is because GL-account classification task requires strong institutional knowledge. It is hard to tell how to book a transaction without referring to historical booking practices.

A.3 Combine Categorical Features with Text Features

Categorical features, such as bank account number, can be concatenated with the transaction description directly in a text format. For example, "the transaction description is <transaction description>. The bank account number is <bank account number>". This would eliminate the need to concatenate text encodings with categorical embedding. However, we did not obtain ideal results with this revised design. With the removal of categorical feature, we observed clear slow-down in the decrease of training loss and increase in oscillation. The experiment was terminated as we are skeptical of the benefits of the revised design.

A.4 Customized Tokenizer

In one experiment, we replaced the pre-trained tokenizer with a customized BPE tokenizer trained using the transaction descriptions. However, with the new tokenizer, it became extremely difficult to train the encoder even after multiple arounds of hyper-parameter adjustments. Since token embedding is the first layer of the language model, embedding values have profound impacts on the final results. We suspect switching tokenizer places the model in an "alien world" where common sense gained in the human world is no longer applicable. As a result, the nature of model training changed from fine-tuning to learn-from-scratch.

B GENERALIZATION TO OUT-OF-SAMPLE DATA POINTS

Due to changing business landscape, new bank accounts could be added and existing booking rules could change. If the model generalized well to out-of-sample data points, we can simply add or adjust labelled data in our embedding databases without fine-tuning our encoder. To verify the generalization capability, we evaluate the encoder by manually adding 50 transactions from bank accounts unseen in training to embedding database without fine-tuning the encoder. The model was able to classify all 50 cases to the correct GL-accounts.

C LONG-RANGE DEPENDENCY ISSUES WITH GENERATIVE LLM

During error analysis, we noticed Alexa-20B tends to fail when the transaction description is long and the correct search string is located at the beginning or in the middle of the description. For example, description is "OTHER REFERENCE: <ID1>RATE=<exchange rate> FX AMT=<amount> CCY=<currency><ID2> ... OGB=<bank name><bank location>...<ID3> ORG=<sender's name> <sender's address> <ID4>....OPI=<sender's account number> <ID5>...<receiver's name>...<IDn><time stamp>". The correct search string, highlighted in orange, is <sender's name> after key word "ORG=". However, Alexa-20B often outputs a wrong entity (e.g. bank name), or even random strings such as "Inc. Inc. Inc". Similar issues are observed on BLOOM-176B, which also tends to output the first bank name presented in the transaction description. To verify the root cause of failures of the generative model, we conducted 3 experiments.

(1) In experiment 1, we truncated all descriptions in the prompt using rules (e.g. truncate to key word "OPI="). Alexa-20B is able to output correct results with the shortened text input.
(2) In experiment 2, we manually move the correct search string to the end of the description. Alexa-20B is able to output correct results.
(3) In experiment 3, we tested AI21 Jurassic-2 Jumbo ([1]), a model with much bigger context window size (8192), with the same sample prompt. We noticed AI21 Jurassic-2 Jumbo is able to generate correct outputs.

Based on those results, we believe the issue we are experiencing are related to long-range dependency.

D SAMPLE PROMPTS USED TO SELECT GENERATIVE LLM

The correct answer is in **bold** and search strings in the demonstrations are highlighted in orange. The task is presented in a "next-word-prediction" format, instead of a question format, because many LLMs we tested have not been instruction-tuned. Prompt 1 follows the format of our search string prompt. For prompt 2, we intend to test the LLM's to perform extraction and generation. We deliberately introduced slight inconsistency in the examples ("Fund Transfer" vs "transfer"). Note that for confidentiality, the examples presented use synthetically generated data but follow the same format as real cash transactions.

Prompt 1

The task is to extract key words from remittance line by following the examples below.
Example 1:
In remittance line "FR:WIRE RECEIPT EU 20231203/REF/AAA SOLUTION INC LUXEBURGER ALLLE, 2319 MULHEIM AN DER RUHR/XION 6D-B126 TRANINV2102-129012-31029, INV-23932-392039091OA JPY:219039 UI232K0D3F9W9DKDl21lJOP21", the correct search string is "6D-B126 TRAN".
Example 2:
In remittance line "FR:WIRE RECEIPT/ EU 20230921/REF/GREAT TECHNOLOGIE CO LTD 23212 AVENUE JOHN F. KENNEDY, NYC PO:2312832/XION 4D-A132 TRAN2323423-34230842, INV40238432-422 1OA EUR:3248048 K078DF3FUW9kDl21X0", the correct search string is "4D-A132 TRAN".
Example 3:
In remittance line "FR:WIRE RECEIPT/ EU 20231201/REF/BEST SANDWITCH GBMH 4293 KURFURSTENDAMM, 342 BERLIN, DEUTSCHLAND/XION 6L-7E TRAN34980248-12931-12, INV-41208013820-128301OA EUR:28103821 E89D57F921F98K211L", the correct search string is "**6L-7E TRANS**"

Prompt 2

The task is to generate general ledger description by following the examples below.
Example 1:
In remittance line “FR:WIRE RECEIPT EU 20231203/REF/AAA SOLUTION INC LUXEBURGER ALLLE, 2319 MULHEIM AN DER RUHR/XION 6D-B126 TRANINV2102-129012-31029, INV-23932-392039091OA JPY:219039 UI232K0D3F9W9DKDl21lJOP21”, correct general ledger description for this line is “6D-B126 Fund Transfer”.
Example 2:
In remittance line “FR:WIRE RECEIPT/ EU 20230921/REF/GREAT TECHNOLOGIE CO LTD 23212 AVENUE JOHN F. KENNEDY, NYC PO:2312832/XION 4D-A132 TRAN2323423-34230842, INV40238432-422 1OA EUR:3248048 K078DF3FUW9kDl21X0”, correct general ledger description for this line is “4D-A132 transfer”.
Example 3:
In remittance line "FR:WIRE RECEIPT/ EU 20231201/REF/BEST SANDWITCH GBMH 4293 KURFURSTENDAMM, 342 BERLIN, DEUTSCHLAND/XION 6L-7E TRAN34980248-12931-12, INV-41208013820-128301OA EUR:28103821 E89D57F921F98K211L", correct general ledger description for this line is "**6L-7E Transfer**" or "**6L-7E Fund Transfer**".

E SYNTHETIC TRANSACTION DATA

We built a synthetic transaction dataset following key-value format. For example, format "/INV0001071178449,ADDITIONAL/AUR1905-110,REFERENCE/ZZF2RLBZ" has keys NaN, ADDITIONAL and REFERENCE. The "/" character is used to separate keys from values and the "," character is used to separate key-value pairs. We deliberately drop some keys, values or delimiter to mimic the data issues observed in real transaction data. Keys are randomly sampled from acronyms frequently seen in cash transactions and values are randomly generated (e.g. IDs) or sampled from public dataset (e.g. corporate name and address). Since most IDs follow certain ID format, we define prefix, length range, composition for IDs to mimic real data. We do not include any payor name (e.g. "MASTERCARD") in our synthetic data because transactions with well-defined payor names tend not to fail CMS, while the focus of our model is to handle ambiguous cases where reusable search strings are not available.
To generate synthetic GL-account, we assume any transactions that share the same keys and delimiters should be booked to the same account. We use rules to generate synthetic search string. If sender’s name is present in description, we select sender’s name; if not, we use reference ID (e.g. AUR41959-4618); if not, we use transaction ID (e.g. O8DL5FSU).
Below are 3 samples from synthetic data.

(1) Description = "OMT=4,750.00 BNF=The EXCELLENT GROUP, INC. ADDITIONAL=AUR41959-1234 COURTHOUSE DRIVE", GL-account = "account 1", search string = "The EXCELLENT GROUP, INC."
(2) Description = "OMT=22000.0 BNF=SMART LEARNING CENTRE AUR0687834-65 555-3909 HUDSON AVE, LAKE CITY", GL-account = "account 1", search string = "SMART LEARNING CENTRE"
(3) Description = "/REFERENCE:O8DL5FSU/OMT:3251182.0/ ADDITIONAL:AUR-467-19013/BNF:CHF-GA-BEACON VISITATION VAL", GL-account = "account 3", search string = "AUR-467-19013"

Example 1 and Example 2 share the same GL-account because both transactions share the same keys and delimiters, even though key "ADDITIONAL" is omitted in Example 2. Example 3 belongs to a different GL-account.

F EXTRACTION SAMPLES FROM GENERATIVE LLM AND BERT

Below are extraction samples from BERT and our extractor. Extracted search strings are highlighted in orange. BERT selects an ID in additional to the ground truth search string. ID that looks like “REF 6008091Q00091989” is often used as search string for a slightly different description format. BERT might not picked up this pattern during training as the number of samples available for each format is rather limited. In addition, there is no written definition of transaction description format.
Inference transaction and ground truth from accountants :
ENTRY-25 JANTRF/REF 6008091Q00091989/ORD/ ACC PAY NATIONWIDE 172223 2I380DJ2M0DHK21/BNF/ AAA SOLUTION BANK BCD CREDIT TRANSFER
KNN retrieved historical transaction 1:
/ENTRY-23 DECTRF/REF 6008476Q0017796/ORD/ON TRACK RETAIL LI 278000 83DSS01M2SS7821 /BNF/ BEST SANDWICH BANK BCD CREDIT TRANSFER
KNN retrieved historical transaction 2:
/ENTRY-30 DECTRF/REF 6008473Q0009621/ORD/ACC PAY NATIONWIDE 172223 2I6SJOP92SDP19FM/BNF/ GREAT PLUMBER BANK BCD CREDIT TRANSFER
Results from generative LLM:
ENTRY-25 JANTRF/REF 6008091Q00091989/ORD/ ACC PAY NATIONWIDE 172223 2I380DJ2M0DHK21/BNF/ GREAT PLUMBER BANK BCD CREDIT TRANSFER
Results from BERT (0.01):
/ENTRY-25 JANTRF/REF 6008091Q00091989/ORD/ ACC PAY NATIONWIDE 172223 2I380DJ2M0DHK21/BNF/ GREAT PLUMBER BANK BCD CREDIT TRANSFER
Results from BERT (0.03):
/ENTRY-25 JANTRF/REF 6008091Q00091989/ORD/ ACC PAY NATIONWIDE 172223 2I380DJ2M0DHK21/BNF/ GREAT PLUMBER BANK BCD CREDIT TRANSFER
Results from BERT (0.05):
/ENTRY-25 JANTRF/REF 6008091Q00091989/ORD/ ACC PAY NATIONWIDE 172223 2I380DJ2M0DHK21/BNF/ GREAT PLUMBER BANK BCD CREDIT TRANSFER